UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Un entorno virtual con clientes remotos sobre la plataforma XNA

Autor: José Miguel Colmena de Celis.

Tutor: José Daniel García Sánchez

Enero 2010

Colmenarejo

Registro de cambios

Versión	Fecha	Razón	Descripción
1.0	6/11/2009	Versión Inicial	Versión inicial del documento.
1.1	11/1/2010	Revisión	Corrección de errores del documento. Añadidos diagramas de secuencia.

Agradecimientos

En primer lugar quiero agradecer a José Daniel García y a Jesús Carretero la confianza que ha depositado en mí para la realización de este y otros proyectos, en los que me han dado la oportunidad de trabajar.

A mis familia, que día a día me animan a que siga adelante, superando todos los escollos que aparecen y que sin duda, sin su apoyo serían muy difíciles de superar.

A Alejandra, mi novia, que ha tenido otros 2 años de paciencia y una buena dosis de templanza para aguantar las largas sesiones de trabajo que exige el estudio de esta carrera.

Finalmente, agradecer a todas aquellas personas, profesores, compañeros y amigos que de una forma o de otra me han permitido aprender de ellos a lo largo de todos estos años.

ÍNDICE

1. Introducción

El proyecto de fin de carrera que se presenta intenta aportar una visión general acerca de los sistemas remotos de generación de escenarios en 3 dimensiones. Para ello se ha desarrollado un prototipo de aplicación con el que realizar una demostración de cómo una serie de personajes puede moverse a través de un entorno tridimensional. Los usuarios que controlan dichos personajes acceden de manera remota al servidor que genera el escenario. Dicho servidor muestra el escenario generado y cómo se mueven los personajes.

Actualmente los sistemas de generación de entornos tridimensionales se ejecutan en los equipos clientes, requiriendo por parte del usuario una fuerte inversión en una máquina capaz de realizar el "Render" de los escenarios y mover los personajes a través de los mismos.

Por el contrario, este proyecto trata de ilustrar como sería un sistema en el que la generación de los entornos reside en el servidor y los clientes solo envían información al servidor acerca de las órdenes que quieren que realice su personaje. Esto posibilita que solo el servidor tenga que generar los modelos, pudiendo emplear máquinas clientes con un limitado rendimiento.

El empleo clientes de reducidas capacidades permite emplear dispositivos tales como teléfonos móviles, PDA o sistemas empotrados a los que se pueden unir dispositivos de geolocalización, de manera que se pueda modelar un entorno real y realizar los movimientos de acuerdo a como se mueve una persona en un entorno real.

Finalmente, gracias al aumento del ancho de banda de las líneas de internet, así como la posibilidad de emitir la imagen generada mediante tecnología de streaming, dicha imagen puede ser enviada al cliente y mostrada en el mismo, posibilitando el empleo de tecnología flash (por ejemplo) o video para hacer llegar al cliente el resultado de las órdenes dadas al sistema.

1.1.Organización del documento

El presente documento muestra el análisis de un prototipo de cómo un sistema puede modelar un entorno en tres dimensiones e ilustrar cómo varios personajes controlados por equipos remotos pueden moverse a través del entorno.

Los dos primeros puntos del documento hacen una introducción al proyecto, así como describir el estado en el que se encuentran las tecnologías empleadas para la realización del mismo, realizando un repaso por los motores de render, servicios web, o tecnologías de streaming.

El tercer punto analiza las necesidades que debe cubrir el sistema, cuáles son los objetivos, el alcance, las tecnologías empleadas y los potenciales usuarios.

El cuarto punto realiza un análisis del sistema mostrando distintos diagramas a nivel de análisis acerca de la aplicación destinada al servidor, los servicios web intermedios entre el servidor y clientes, y finalmente los distintos clientes disponibles para la realización de pruebas.

Finalmente, en los últimos puntos, se muestra un manual de usuario del sistema, algunas conclusiones sobre el presente proyecto así como posibles líneas de expansión para el futuro y una serie de anexos y referencias.

2. Estado del Arte

En el siguiente apartado se tratará el estado en el que se encuentran actualmente las distintas tecnologías empleadas en este proyecto.

Entre estas tecnologías destacamos los motores de render, servicios web o los servicios de streaming que están tan de moda últimamente.

2.1.Motores de render

Se denomina Rendering o Renderizado al proceso de generar una imagen a partir de un modelo usando un programa de ordenador. Este programa puede tener diversas utilidades pero la principal es el Motor de Render, responsable de generar la imagen, con sus luces, transparencias, efectos y objetos del modelo.

Entre las capacidades que un motor de render debe tener podemos destacar las siguientes:

- Shading: es el proceso de calcular como varía el color y brillo en función de la luz
- Texture Mapping: es el proceso para aplicar detalles a superficies.
- Bump Mapping: es el proceso para aplicar relieve a una superficie en función de una textura.
- Shadows: es el proceso para calcular proyecciones de los objetos para generar sombras, sobre este proceso se apoya el de soft shadows que permite variar el color de la sombra en función de la iluminación.
- Transparency y Translucency: es el proceso para generar superficies transparentes o translucidas.

- Motion Blur: es el proceso para calcular como se difuminan los objetos debido a altas velocidades en los movimientos.

Existen otras capacidades que los motores de render implementan, principalmente encaminadas a añadir realismo a las imágenes o para añadir efectos de cámara cinematográfica.

Todas estas capacidades antes descritas se dan gracias al empleo de diferentes técnicas aplicadas al modelo. Estas técnicas suponen un coste computacional muy grande, ya que realizan gran cantidad de cálculos sobre cada pixel de la imagen final.

Entre estas técnicas destacaremos las siguientes:

- Ray casting: usado principalmente para simulaciones en tiempo real, como puede ser en videojuegos o dibujos animados donde lo más importante no es el detalle sino el rendimiento. La imagen se genera pixel a pixel como si los rayos de luz partieran del punto de visión del usuario. Mediante esta técnica los colores aplicados a un pixel se toman del modelo o bien de las texturas del mismo directamente, aunque hay la opción de que dicho color sea calculado en función de la iluminación. Para reducir el coste computacional del renderizado se puede reducir el número de rayos de luz.
- Radiosity: este método pretende simular la forma en la que los objetos directamente iluminados actúan a modo de iluminación indirecta a otros objetos. Esta técnica produce imágenes con sombreados realistas así como realiza una mejor captura del ambiente en una escena de interior. Debido a la naturaleza Recursiva/Iterativa del algoritmo empleado, la simulación de objetos complejos se hace muy lenta y complicada.

- Ray tracing: la técnica de Ray tracing pretende simular la luz como si fueran partículas. Para la aplicación de esta técnica se aplican diferentes métodos, pero en la mayoría de las implementaciones se permite a la luz propagarse en líneas rectas. Ray tracing es una técnica que se aplica mediante fuerza bruta por lo que se hace imposible aplicarlo a sistemas de renderizado en tiempo real, incluso se hace demasiado lento para la generación de pequeñas películas. Sin embargo, se están realizando grandes esfuerzos para reducir la cantidad de cálculos necesarios para determinados píxeles que no requieren demasiado detalle que no dependen del de las características del Ray tracing. Así mismo existen implementaciones hardware de esta técnica, aunque aún en fase de prototipo, que pretende acelerar el proceso.

En el mercado existen muchos motores de render dependiendo del objetivo del mismo, motores para videojuegos, para generación de imagen realista, para generación de video.

Como ejemplo de motores de render podemos citar:

Ogre: Ogre es un Motor de render Open Source destinado a videojuegos. Entre sus principales características figuran la carga de texturas de diferentes formatos, animación por esqueletos, sistemas de partículas. Desde el punto de vista del desarrollo, Ogre expone un framework completamente orientado a objetos así como soporte para la api de Direct 3d y OpenGL.

V-Ray: es un motor de render usado como extensión para algunos programas de modelado 3D. V-Ray emplea técnicas avanzadas como pueden ser algoritmos de iluminación global, tales como Path Tracing o Photon Mapping. Comparado con los motores de render

incluidos en los sistemas de diseño 3D, V-Ray genera imágenes más realistas, sobre todo aplicado a proyectos de arquitectura.

2.1.1. Shading

Shading es el proceso usado en dibujo para representar niveles de oscuridad aplicando más densidad o con una sombra más oscura para áreas más oscuras, o menor densidad o sombras mas difuminadas para áreas más luminosas.

Gracias a las técnicas de *Shading* se permite dar a los objetos la sensación de profundidad de la que carecen al representarlos en una pantalla en 2 dimensiones.

Aplicado a gráficos por ordenador, *Shading* se refiere al proceso de alterar el color basándose en el ángulo de la luz y la distancia de la luz respecto del objeto. Los efectos de *Shading* se generan durante el proceso de render.

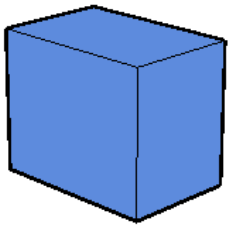
Los efectos de *Shading* alteran los colores de las caras en un modelo 3D basándose en el angulo de la superficie respecto a las fuentes de luz del modelo.

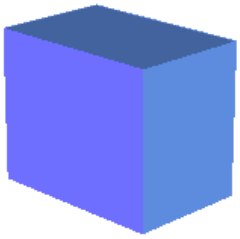
En la primera imagen se ven las caras de la figura renderizada, pero todas del mismo color. Se han añadido las aristas de las caras para facilitar la visualización de las mismas.

La segunda imagen es el mismo modelo al que se le han eliminado las aristas. Se hace complicado determinar dónde está cada una de las caras.

En la tercera imagen se ha realizado el render aplicando los efectos de *Shading*, haciendo que la imagen parezca mucho más realista, permitiendo ver mejor cada una de las caras.

Ejemplo:

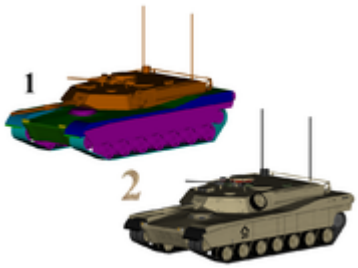




Render de una caja sin efectos de *Shading*, pero empleando aristas para separar las caras.	**Misma caja a la que se le han eliminado las aristas.**	**Misma imagen renderizada aplicando *Shading*.**

2.1.2. Texture Mapping

Texture mapping es un método para añadir detalles, texturas o colores a un gráfico generado por ordenador o un modelo 3D.



Modelo 3D sin texturas y con texturas.

Una textura es aplicada (o mapeada) sobre la superficie de un polígono. Mediante esta técnica es posible aplicar más de una textura sobre una superficie, se pueden aplicar texturas de luces para iluminar un objeto como alternativa a recalcular las luces en esa parte del objeto, cada vez que es renderizado, ahorrando recursos computacionales.

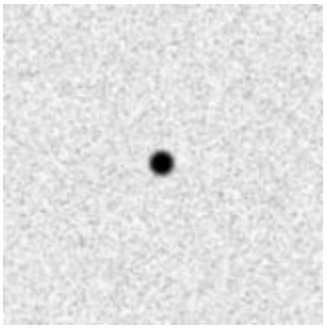
Al aplicar texturas es posible que sea necesario aplicar correcciones. Las coordenadas se especifican para cada vértice de un triangulo dado, después, estas coordenadas son interpoladas aplicando el algoritmo de líneas de Bresenham. Si las coordenadas de dicha textura son interpoladas linealmente en la pantalla, el resultando es el mapeado afín de una textura. Está es una forma rápida de calcular el mapeado de una textura, pero se pueden apreciar discontinuidades entre triángulos adyacentes cuando.

Cuando se aplica una textura a un objeto, se aprecian defectos en los polígonos cuando estos no están perpendiculares a la vista.

2.1.3. Bump mapping

Bump mapping es una técnica para gráficos generados por ordenador, donde a cada pixel se aplica una perturbación a la normal de la superficie del objeto respecto a una textura dada y se aplica antes de los efectos de iluminación. El resultado es una superficie más detallada, representando mejor los efectos del mundo natural. Las formas más empleadas en *Bump mapping* son Normal y Parallax mapping, que emplean una escala de grises para aplicar los efectos.

Esfera sin aplicar *bump mapping*



Textura para aplicar *bump mapping*

Esfera una vez aplicada la textura anterior con los efectos de *bump mapping*

Para determinar como de fuerte va a ser la deformación aplicada a la superficie debido al *Bump Mapping* se necesita calcular la normal de dicha superficie, está define como interactúa el objeto cuando recibe la luz, cuanto más perpendicular es la luz al objeto menos se aplicará la deformación. El resultado es una superficie que parece tener una profundidad real. Además el algoritmo asegura que cambia la apariencia de la superficie según se mueven las luces por la escena.

2.1.4. Motion blur

Motion blur o "trazo confuso de movimiento" es el aparente movimiento a gran velocidad de objetos en una imagen quieta (fotografía) o una secuencia de imágenes como una película o animación.

En la animación en por ordenador, cada marco muestra una imagen estática con el tiempo (análogo para una cámara con un obturador infinitamente rápido), con *trazo confuso* de movimiento de cero. Esto provocará en un videojuego con una tasa de refresco de 25-30 cuadros por segundo, que el movimiento se vea poco natural, mientras el movimiento natural filmado en la misma tasa del marco aparece continuo. Para compensar para esto, las tasas del marco se utilizan tasas de refresco más alta, de 60 marcos por segundo o más, que simulan dicho efecto. La mayoría de los videojuegos de la siguiente generación utilizan *trazo confuso de movimiento*, especialmente los de carreras o simuladores de vuelo, para simular una velocidad acelerada (como una cámara).

Ejemplo:

De izquierda a derecha, imagen original, imagen difusa, imagen con difuminación compensada.

2.1.5. Ray casting

Ray casting es un algoritmo empleado en generación de imágenes tridimensionales por ordenador destinadas a una pantalla en dos dimensiones. Para ello se simula el trazado de rayos desde el ojo del observador a una fuente de luz. Esto elimina la posibilidad de obtener reflejos, refracciones o la difuminación natural de sombras en los objetos. Debido a que los cálculos son muy rápidos, el algoritmo de *Ray casting* se empleaba en los prmieros sistemas de render en tiempo real para Video Juegos.

Una fuente de luz emite un rayo de luz que viaje hasta que una superficie se cruza en su trayectoria. En el momento en el rayo se cruza con la superficie puede ser absorbido, reflejado o refractado. Una superficie puede reflejar toda o una parte de un rayo de luz, en una o más direcciones, igualmente, puede absorber solo parte de un rayo de luz, resultando en una pérdida de intensidad de la reflexión o la refracción. Si la superficie tiene algún efecto de transparencia o translucencia, esta refracta una porción del rayo hacia sí misma.

El primer algoritmo de *Ray casting* para renderizados fue presentado en 1968. La idea detrás del *Ray casting* es lanzar rayos desde el ojo del observador hasta cada pixel y encontrar los objetos más cercanos que bloquean el camino de dicho rayo. Supongamos que se dispone una imagen en la pantalla donde cada cuadro de la imagen es un pixel en la pantalla. Normalmente el ojo ve cada pixel. Usando las propiedades del material y el efecto de las luces en la escena, el algoritmo determina los efectos de *Shading* de dicho objeto. Los *Shadings* de la superfice son calculados usando un modelo tradicional de *Shading* de gráficos generados por ordenador.

Una ventaja importante que ofrece el algoritmo de *Ray casting* es la habilidad de trabajar sobre superficies que no son planas y con sólidos, como conos o esferas. Si una

superficie matemática puede ser intersecada por un rayo, esta puede renderizarse usando *Ray casting*.

2.1.6. Ray tracing

El *ray tracing*o trazado de rayos es otro de los algoritmos empleados para la generación de imágenes en tres dimensiones. Está basado en el algoritmo de *Ray casting* que se ha explicado anteriormente.

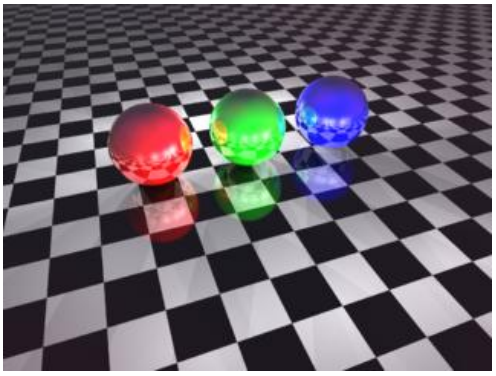
El algoritmo de trazado de rayos extiende la idea de trazar los rayos para determinar las superficies visibles con un proceso de sombreado (cálculo de la intensidad del píxel) que tiene en cuenta efectos globales de iluminación como pueden ser reflexiones, refracciones o sombras arrojadas.

Para simular los efectos de reflexión y refracción se trazan rayos recursivamente desde el punto de intersección que se está sombreando dependiendo de las características del material del objeto intersecado.

Para simular las sombras arrojadas se lanzan rayos desde el punto de intersección hasta las fuentes de luz. Estos rayos se conocen con el nombre de rayos de sombra.

En la actualidad, el algoritmo de trazado de rayos es la base de otros algoritmos más complejos para síntesis de imágenes que son capaces de simular efectos de iluminación global complejos como la mezcla de colores (color bleeding).

Ejemplo:



Tres esferas a las que se les a aplicado *Ray casting*, reflejándose en el suelo y cada una en sus esferas adyacentes.

El algoritmo para realizar *Ray casting* es el siguiente:

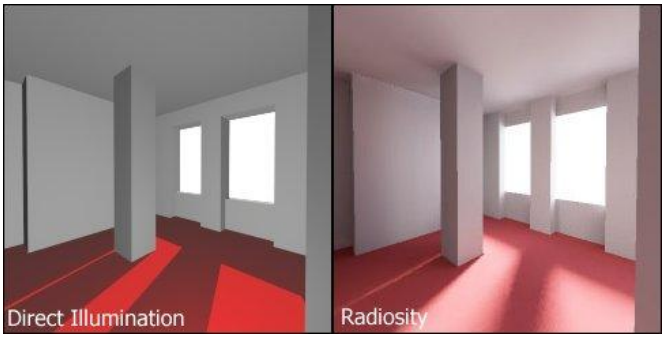
```
Para cada pixel de la imagen{
 Crear un rayo desde el punto de visión a través del pixelActual
 Inicializar NearestT al INFINITO y NearestObject a NULL
 Para cada objeto de la escena {
   Si el rayo intercepta el objetoActual{
     Si t de la intersección es menor que NearestT {
         Poner NearestT = t de la intersección
         Poner NearestObject a objetoActual
     }
   }
 }
 Si NearestObject = NULL{
    Rellenamos pixelActual con el color de fondo
 }
 Sino{
    Lanzar un rayo a cada foco de luz para comprobar las sombras
    Si la superficie es reflectiva, generar un rayo reflectivo
(recursivo)
    Si la superficie es transparente, generar un rayo refractante
(recursivo)
    Usar NearestObject y NearestT para computar la función de
sombreado
    Rellenar este pixel con el color resultante de la función de
sombreado
 }
}
```

2.1.7. Radiosity

Radiosity es un algoritmo de iluminación global usando en la generación de gráficos 3D por ordenador. *Radiosity* es una aplicación del método el elemento finito para resolver la ecuación de render para escenas con superficies difusas. *Radiosity* tiene en cuenta fuentes de luz difusas que son reflejadas un número de veces antes de alcanzar el ojo del observador.

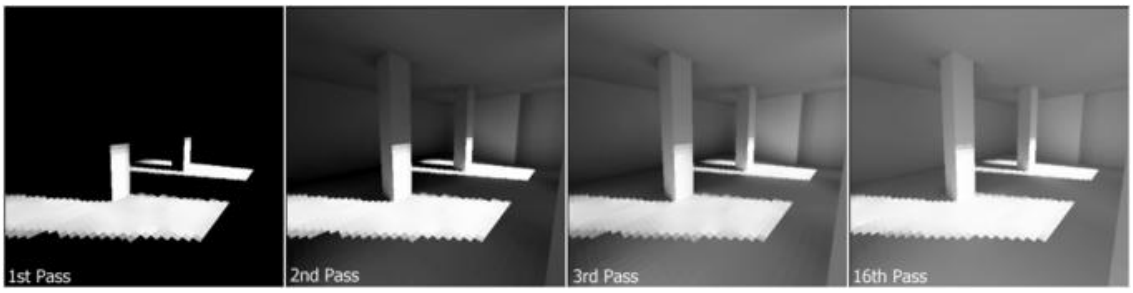
Los cálculos de *Radiosity* son dependientes del punto de vista del observador, lo cual incrementa enormemente los cálculos a realizar, pero es muy útil para cualquier punto de vista.

La inclusión de cálculos de radiosidad en el proceso de renderizado permite añadir un elemento de realismo al resultado final de la escena. Supongamos una habitación simple, la imagen de la izquierda ha sido renderizada con un algoritmo típico de iluminacion. En la escena hay tres tipos de iluminación colocados en puntos específicos con el objetivo de crear una imagen lo más realista posible. Como se ve el resultado en las paredes no parece nada realista. La imagen de la derecha fue generada aplicando algoritmos de radiosidad. Solo hay una fuente de luz, exterior a la habitación. Las sombras se ven difuminadas en las superficies, afectadas por los reflejos de la luz en las superficies. Además, el color rojo del suelo, se ve reflejado en las paredes. Estos efectos no fueron elegidos por el diseñador de la habitación, sino que aparecen al aplicar radiosidad.



Las superficies de la escena que va a ser renderizada son divididas cada una en una o más superficies menores.

Los algoritmos de radiosidad resuelven la ecuación de renderizado de manera iterativa, aplicando a cada iteración valores de radiosidad intermedios. Dichos valores corresponden a distintos niveles de rebote de las luces. Tras una primera iteración, vemos como la escena se ve con un solo nivel de rebote, tras la segunda pasada, con dos y así sucesivamente. La radiosidad progresiva es útil para obtener una vista previoa de la escena.



Escena generada con *Radiosity*. Se observa como la imagen se va clarificando a cada pasada del algortimo.

Una de las ventajas de los algoritmos de radiosidad es que es relativamente simple de implementar. Esto hace que sea un algoritmo útil para la enseñanza acerca de los algoritmos de iluminación global. No requiere de un amplio conocimiento matemático para entender o implementar este algoritmo.

La principal desventaja que presentan los algoritmos de radiosidad son los problemas que tienen para resolver cambios repentinos respecto del punto de vista. Ya que hay que volver a generar toda la imagen.

2.2.XNA

XNA es un conjunto de herramientas con un entorno de ejecución manejado creado por Microsoft. XNA facilita el proceso de desarrollo y mantenimiento de un Videojuego, evitando que los programadores tengan que escribir código que se repite en varios puntos del programa.

La plataforma XNA fue publicada por primera vez en 2006, con el lanzamiento de la versión 1.0 junto con un IDE de desarrollo similar a Visual Studio, XNA Game Studio. La versión actual es la 3.1, permitiendo integrarse completamente con la versión 2008 de Visual Studio. Así mismo, XNA permite el desarrollo y mantenimiento de videojuegos tanto para PC como para XBOX 360 o Microsoft Zune, realizado unos cambios mínimos, gracias a la máquina virtual que ejecuta .Net.

2.2.1. XNA Framework

El framework de XNA está basado en la implementación del .Net Compact Framework en el caso de XBOX 360 o Zune y en la versión completa .Net Framework en el caso de aplicaciones para PC.

Este framework se ejecuta cobre una versión CLR (Common Languaje Runtime) optimizada para el desarrollo y gestión de videojuegos. El CLR está disponible tanto para Windows XP, Vista y Windows 7, así como para XBOX 360. Ya que los juegos se ejecutan sobre este CLR, pueden ser ejecutados en cualquier otra plataforma que soporte el framework de XNA realizando unos cambios mínimos en el código escrito.

Gracias al CLR los programas escritos con la plataforma XNA pueden ser escritos en cualquier lenguaje que soporte la especificación de .Net, pero solo C#, XNA Game Studio y todas las versiones de Visual Studio están soportadas oficialmente por Microsoft.

2.2.2. Motor de render en XNA

Una de las grandes ventajas de la plataforma XNA es que es totalmente orientada a componentes.

Gracias a esta orientación a componentes, admite el empleo de diferentes motores de render de terceras compañías. Aún así, el framework incluye su propio motor de render que incluye técnicas para aplicar shaders, shadows, fogging y otras de las que se han descrito anteriormente.

Así mismo, permite la aplicación de efectos de luz gracias a un lenguaje de definición de efectos llamado HSLS (High Level Shader Languaje) que interactúa directamente sobre la api de DirectX.

2.3.Servidores

El término servidor, en el campo de la informática, está referido a un ordenador, que conectado a una red, provee de servicios al resto de ordenadores de la red.

Así mismo, también se entiende como servidor a un programa informático que está destinado a ofrecer un servicio a otros programas informáticos que se ejecutan en la misma máquina o en otras máquinas de la red.

Gracias a estas aplicaciones servidoras, se implementaron soluciones bajo la arquitectura Cliente-Servidor, pero esto no implica que bajo esta arquitectura se requieran dos o más máquinas.

Actualmente existen multitud de aplicaciones y máquinas que actúan como servidores, entre los más empleados podríamos citar los siguientes:

- Servidores web: servidores que proveen acceso a recursos web como son paginas HTML, imágenes o archivos bajo el protocolo HTTP.
- Servidores de correo: servidores que almacenan correos electrónicos, estos se dividen en dos grupos los dedicados al envío y los dedicados a la recepción, se apoyan sobre los protocolos SMTP para envío y POP o IMAP para recepción.
- Servidores de bases de datos: servidores que se encargan de proveer acceso a bases de datos.
- Servidores DNS: son los servidores encargados de traducir los nombres de dominio a direcciones IP.

Existen muchas otras aplicaciones servidoras que no se describirán en este documento ya que no tienen influencia en la realización del presente proyecto.

2.4.Servicios Web

Un servicio Web[8] es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios Web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

La principal razón para usar servicios Web es que se basan en HTTP sobre TCP (*Transmission Control Protocol*) en el puerto 80. Dado que las organizaciones protegen sus redes mediante *firewalls* -que filtran y bloquean gran parte del tráfico de Internet-, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web se vehiculan por este puerto, por la simple razón de que no resultan bloqueados.

Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran *ad-hoc* y poco conocidas, tales como EDI *(Electronic Data Interchange*), RPC, u otras *Application Programming Interface*, APIs.

Una tercera razón por la que los servicios Web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad

será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más acusada.

Los Servicios Web proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

Ejemplo de funcionamiento de un Servicio Web.

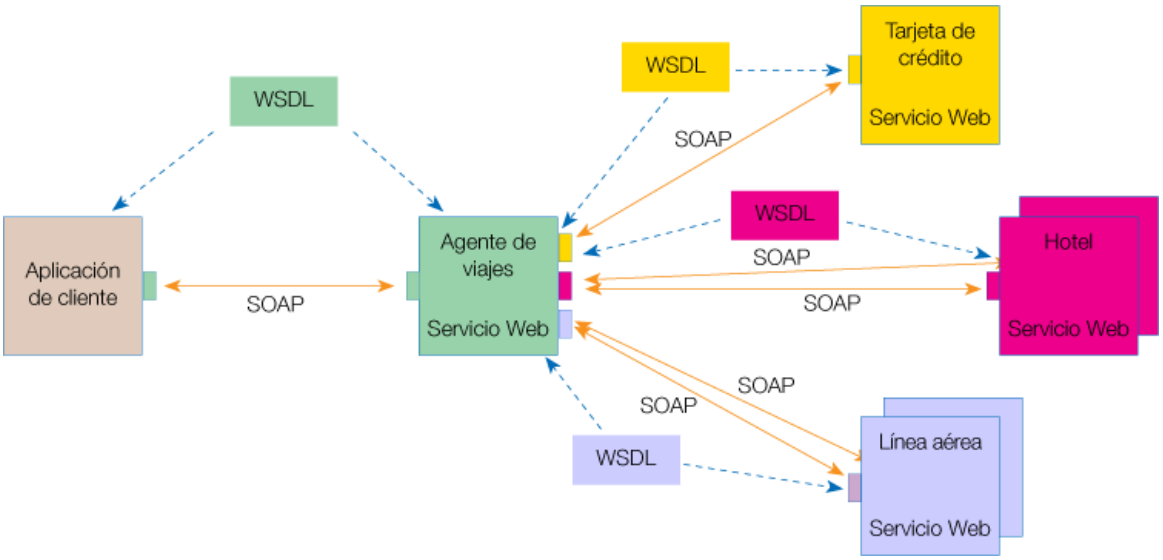


Figura 3 - Los servicios Web en Funcionamiento

Según el ejemplo del gráfico, un usuario (que juega el papel de cliente dentro de los Servicios Web), a través de una aplicación, solicita información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes que ofrece sus **servicios** a través de

Internet. La agencia de viajes ofrecerá a su cliente (usuario) la información requerida. Para proporcionar al cliente la información que necesita, esta agencia de viajes solicita a su vez información a otros recursos (otros Servicios Web) en relación con el hotel y la línea aérea. La agencia de viajes obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros Servicios Web que le van a proporcionar la información solicitada sobre el hotel y la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia de viajes que servirá de intermediario entre el usuario y el servicio Web que gestionará el pago.

En todo este proceso intervienen una serie de tecnologías que hacen posible esta circulación de información. Por un lado, estaría SOAP (Protocolo Simple de Acceso a Objetos). Se trata de un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, etc. SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un *envelope* (sobre), cuya estructura está formada por los siguientes elementos: *header* (cabecera) y *body* (cuerpo).

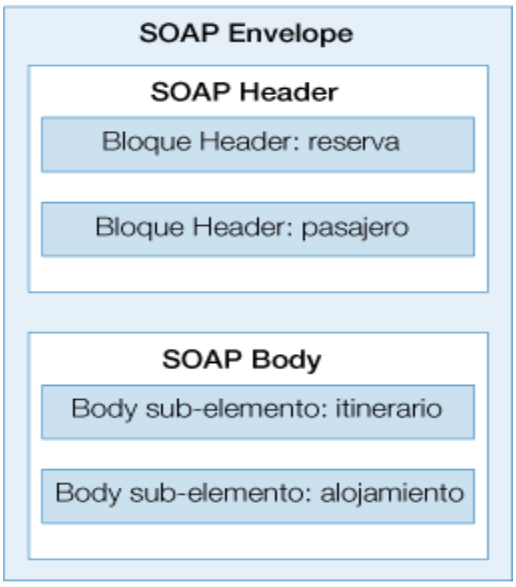


Figura 4 - Estructura de los mensajes

Por otro lado, WSDL (Lenguaje de Descripción de Servicios Web), permite que un servicio y un cliente establezcan un acuerdo en lo que se refiere a los detalles de transporte de mensajes y su contenido, a través de un documento procesable por dispositivos. WSDL representa una especie de contrato entre el proveedor y el que solicita. WSDL especifica la sintaxis y los mecanismos de intercambio de mensajes.

2.5.Streaming

Streaming es un término que se refiere a la reproducción bajo demanda de un contenido multimedia a través de una red sin necesidad de realizar una descarga previa, dicha red puede ser internet, una red local o cualquier otra red de comunicaciones. Esto permite ver programas de televisión o escuchar la radio a través de internet, así como la difusión de otros contenidos multimedia como pueden ser videojuegos, música, etc…

La tecnología streaming como tal apareció en 1995 con el lanzamiento de Real Audio 1.0. Antes de su lanzamiento, la reproducción de contenido multimedia requería de la descarga completa del archivo. Puesto que estos archivos son bastante grandes, del orden de varios MegaBytes, incluso GigaBytes, la reproducción de los mismos requería de un tiempo de espera muy elevado. Sin embargo, gracias a la tecnología de streaming, se puede descargar y reproducir de manera simultánea.

La tecnología de streaming se sustenta sobre cuatro pilares fundamentales:

- Codecs: son un conjunto de programas que permiten la decodificación de los ficheros multimedia.
- Protocolos de transporte ligeros: los datos se transfieren mediante el protocolo de transporte UDP y RTSP (Real Time Streaming Protocol), esto hace que la entrega de

paquetes hacia el cliente se haga de una manera más eficaz, gracias al modo de funcionamiento de UDP que favorece el flujo continuo de paquetes de datos.

- Precarga de contenidos: para evitar retrasos o paradas en la reproducción cuando los datos escasean, ya sea por cortes de conexión o por saturación de la red, los reproductores multimedia almacenan en un buffer parte del contenido que deben reproducir a medida que lo van recibiendo.
- Red de distribución de contenido: cuando un recurso empieza a ser demandado por demasiados usuarios, se pueden producir retrasos en la reproducción, por lo que se hace necesaria una red de distribución, que no es más que una red de una organización o varias, que se unen para proveer ancho de banda para la distribución de dicho contenido.

3. Objetivos del proyecto

En el siguiente apartado se tratarán los objetivos del proyecto realizado, qué debe hacer y qué no debe hacer. Se realizará una descripción de las aplicaciones que se han implementado así como de los potenciales usuarios a los que va destinada. Finalmente se describirá el entorno de ejecución del sistema, así como las tecnologías implicadas en el mismo.

3.1.Propósito

El propósito del proyecto que se ha realizado es obtener un entorno de representación de entornos virtuales a través de modelos generados previamente. La generación de los modelos se realiza en un servidor que está a la espera de que los clientes le envíen órdenes para que sean representadas a través de personajes.

El programa que trabaja a modo de servidor se encarga de escuchar las peticiones que le llegan desde la red. Este proceso crea un hilo de ejecución que se encarga de arrancar el motor de XNA para la representación de los entornos y personajes.

Una vez que el servidor recibe una petición de servicio de un cliente, extrae la orden que viaja en la petición y la almacena en una cola de órdenes. Esto permite que las peticiones se traten de una manera rápida, permitiendo el uso del servidor por parte de varios clientes de manera simultánea.

Para permitir la comunicación entre clientes y servidor, disponemos de un web service intermedio que es el responsable de recibir las peticiones de los clientes y reenviarlas al servidor. Aunque esta es la forma de trabajo por defecto, el servidor también admite que los

clientes se conecten directamente a él, pero tiene una restricción, trabajando de esta manera, solo admite un cliente, lo cual limita el funcionamiento del sistema.

Esta arquitectura permite tratar la concurrencia de clientes en una capa intermedia repersentada por unvweb service, de manera que toda la problemática del control de concurrencia en el servidor se traslada a la capa que ejecuta el web service. Así mismo, ya que el web service requiere de un servidor web, esto permite una alta escalabilidad, gracias a la escalabilidad que permiten los servidores web actuales.

En cuanto a los clientes, se han desarrollado varios prototipos de clientes que se comunican con el web service. El objetivo principal de los clientes es poder leer las órdenes desde un fichero y enviárselas al servidor. Para ello tenemos un cliente que abre un fichero de órdenes e invoca el método remoto del web service que corresponda a la orden a enviar.

Así mismo disponemos de otro cliente que en lugar de leer un fichero de órdenes se encarga de generarlo. Para ello puede o bien invocar los métodos del web service o invocar directamente al motor de XNA y enviarle las ordenes directamente. Mediante el uso de este cliente podemos ver la ruta que va a seguir el personaje, viendo el resultado final del fichero que se va a generar. Cada vez que enviamos una orden, está se añade a una lista para que al final podamos salvarla en un fichero de texto.

Finalmente, disponemos de una capa intermedia para comunicar los clientes con el web service. Esta capa permite unificar todas las peticiones que podrían enviar los distintos clientes y se encargaría de realizar el envío al web service.

Esta biblioteca facilita la ampliación del sistema por parte de futuros clientes, así como el desarrollo de nuevos clientes.

3.2.Alcance

En este apartado se va a tratar los requisitos que debe implementar o no el sistema que se presenta en este proyecto. Se describirán por un lado los requisitos funcionales, que indican que es lo que va a implementar el sistema. Por otro lado, se describirán los requisitos inversos, que son los que nos dicen que es lo que no va a hacer este prototipo.

3.2.1. Descripción de requisitos funcionales.

A continuación se van a detallar las funcionalidades principales que debe implementar el sistema completo, describiendo cuales se implementaran en los clientes y cuales en los servidores.

La aplicación que se ejecuta en el servidor debe ser capaz de generar un entorno en 3D ya creado con una herramienta de modelado. Así mismo debe ser capaz de extraer órdenes de una cola de órdenes para mover personajes por el entorno.

El sistema será capaz de representar distintos personajes, que se mueven por el entorno en base a las órdenes que se les da. El sistema es el responsable de detectar las colisiones que se puedan producir entre los personajes y el entorno o entre los propios personajes, de manera que cuando un personaje colisiona, deja de moverse.

En cuanto al web service, será capaz de recibir las siguientes órdenes: crear un personaje, borrar un personaje, mover un personaje, rotar un personaje, mover un personaje de un punto a otro mediante una trayectoria, obtener la posición actual de un personaje, saber si existe un personaje, cambiar de cámara y establecer un parámetro para permitir que las cámaras vayan cambiando cada cierto tiempo.

Finalmente los clientes serán capaces de, por un lado, abrir un fichero de texto que contiene las órdenes que se van a enviar, leer dichas órdenes e invocar el método del web service adecuado a la orden leída. Por otra parte, también deben ser capaces de generar el fichero de órdenes mostrando el entorno con el personaje que va a seguir dicha ruta. Finalmente, este cliente deberá añadir las órdenes a una lista para posteriormente poder guardar en disco un fichero de texto que contenga las órdenes.

3.2.2. Descripción de requisitos inversos.

Para esta primera versión del proyecto se han impuesto algunas limitaciones, sobre todo en la aplicación destinada al servidor. Estas limitaciones quedarán reflejadas en los requisitos inversos, describiendo que es lo que no va a hacer cada una de las partes del sistema.

La aplicación del servidor no implementará animaciones en los personajes. Las animaciones en personajes suponen hacer diseños más detallados de los modelos y no es el objetivo del proyecto. Tampoco se podrá ejecutar en sistemas XBOX 360 debido a la arquitectura elegida para las comunicaciones entre el programa servidor y el sistema de renderizado de modelos.

En cuanto a los movimientos que puede representar el sistema de renderizado, no están incluidos movimientos por trayectorias que contengan puntos intermedios, es decir, actualmente los movimientos por trayectoria se efectúan desde el punto en el que se encuentra el personaje hasta un punto de destino, pero no puede indicársele que vaya de un punto inicial a uno final pasando por algunos intermedios. Tampoco permite decirle a un personaje que vaya a una habitación en concreto.

Finalmente los clientes no podrán generar ficheros con un formato diferente a .TXT, así como no podrán leer de otros formatos distintos al mencionado. Tampoco serán capaces de iniciar la aplicación por sí mismos, por lo que ésta debe estar iniciada en el servidor correspondiente. Así mismo, los clientes no podrán comunicarse con el servidor de ninguna manera que no sea a través del web service habilitado para ello, como por ejemplo por sockets directamente u otros mecanismos software de comunicación de procesos.

3.3.El prototipo de aplicación

En este apartado se describirán las funcionalidades implementadas en el prototipo de aplicación realizado para este proyecto.

3.3.1. Funciones del cliente

Los prototipos del cliente que se han implementado se encargan por un lado de obtener los datos para generar un avatar, necesitando únicamente el nombre del avatar, ya que el modelo a representar se indica en el fichero de órdenes. También abrirá una ventana en la que seleccionar el fichero de órdenes del cual se leerán las órdenes. Finalmente iniciará la lectura del fichero y ejecutará los métodos remotos del web service adecuados a cada orden.

Por otro lado, el segundo cliente permite o bien abrir una ventana en la que ver el renderizado del modelo o bien enviar las ordenes directamente al web service. El cliente recoge el nombre del personaje y el modelo a emplear. Como órdenes que puede enviar están la de creación y destrucción del modelo, cualquier tipo de movimiento, rotaciones y movimiento por trayectorias, para ello se abre una nueva ventana en la que especificar el punto de destino. Este cliente también dispone de una funcionalidad para poder cambiar la cámara activa así como indicar un valor que determina el intervalo de tiempo para que se cambien las cámaras de una a otra de manera automática.

3.3.2. Funciones del Web Service

El web service implementa los métodos remotos para enviar al servidor órdenes de movimiento, realizar rotaciones, crear y eliminar el personaje, realizar un movimiento por trayectoria, obtener la posición actual, saber si un personaje existe en el sistema, cambiar a la siguiente cámara o establecer el temporizador para el cambio automático de las cámaras.

3.3.3. Funciones del servidor

El servidor consta de una consola y una ventana donde se muestra el modelo y los personajes. La aplicación de consola abre una conexión y se mantiene a la escucha de peticiones que le lleguen por el puerto 15000 con las órdenes dadas a cada personaje. Una vez que recibe una orden, la envía al sistema de renderizado y éste se responsabiliza de ejecutar las acciones adecuadas a dicha orden.

El sistema de renderizado, además de procesar las órdenes, como ya se ha comentado en el párrafo anterior, muestra los entornos gráficos así como los personajes. También se encarga de calcular los puntos intermedios en los movimientos por trayectorias y gestiona las colisiones entre los distintos personajes y el entorno.

3.4.Los usuarios

Actualmente no se tiene un tipo de usuario definido para el sistema propuesto. En una primera versión estará dirigido a personal investigador con formación técnica en sistemas informáticos.

Dentro de lo previsto, los futuros potenciales usuarios del sistema podría ser personal de seguridad, que utilizan el sistema para monitorizar entornos vigilados, empresas de simulación de comportamiento de personas en entornos virtuales.

Otro de los usos que se contemplan para futuras versiones del sistema es la generación automática de entornos basándose en ficheros de texto que contienen la parametrización de un entorno. Los posibles usuarios de esta evolución del sistema podrían ser usuarios de aplicaciones de mapas que representen edificios en 3 dimensiones, como puede ser google maps.

3.5.Entorno y tecnología

A continuación se describen las distintas tecnologías que han sido empleadas para la realización de este proyecto.

3.5.1. Tecnología hardware

La tecnología hardware empleada ha sido elegida en base a los equipos en los que se han realizado las pruebas, el desarrollo y que los terminales empleados son comunes en las organizaciones, a las cuales va dirigida la aplicación.

3.5.1.1. Hardware en los clientes

El hardware necesario para estos terminales será un equipo dotado con procesador *Intel* o *Amd*, de no menos de 3 *Gigahertzios*, así como 2 *Gigabytes* de memoria RAM y una capacidad de disco duro de 100 *Gigabytes*. Todos irán equipados con teclado y ratón inalámbrico.

Así mismo las estaciones cliente requerirán de hardware para conexión a la red, preferiblemente a internet, pero si el servidor reside en la red local bastará con una conexión Ethernet de 100Mb.

Finalmente, deberá tener una tarjeta gráfica compatible con directX 9 y pixel shader 1.0 para poder ejecutar el motor de XNA 1.1.

3.5.1.2. Hardware en el servidor

Para el despliegue de la aplicación en el servidor se han hecho pruebas en una máquina con un procesador dual core a 2,53 Ghz con 4 Gb de Ram.

Al igual que los clientes deberá disponer de hardware de conexión a la red, siendo de 1Gbps para redes Ethernet, así como una tarjeta gráfica que soporte DirectX 9 y pixel shader 1.1.

3.5.2. Tecnología Software

El software elegido para la realización de este proyecto ha sido la plataforma Windows ya que es la más empleada dentro de las organizaciones objetivo del sistema.

3.5.2.1. Software en los clientes

Las máquinas clientes deberán emplear Windows XP en cualquiera de sus versiones o superiores. Puesto que el sistema está desarrollado bajo la plataforma .Net deberá tener

instalado el conjunto de librerías .Net Framework 2.0, así como las librerías del framework de XNA.

3.5.2.2. Software en el servidor

El servidor deberá ejecutar Windows XP o superior, siendo recomendable Windows server 2003 o superiores. Tendrá que tener instalado el servidor web IIS 6 como mínimo, así como tener instalado el conjunto de librerías .Net Framework 2.0, así como las librerías del framework de XNA.

4. Análisis del sistema

A continuación se da una visión de la fase de análisis del sistema. En esta fase se ha llevado a cabo un estudio de las distintas arquitecturas posibles para la aplicación, estudiando ventajas y desventajas de los distintos sistemas de comunicaciones, así como la realización de un análisis mediante modelos de casos de uso y modelos de clases de análisis empleando el lenguaje UML.

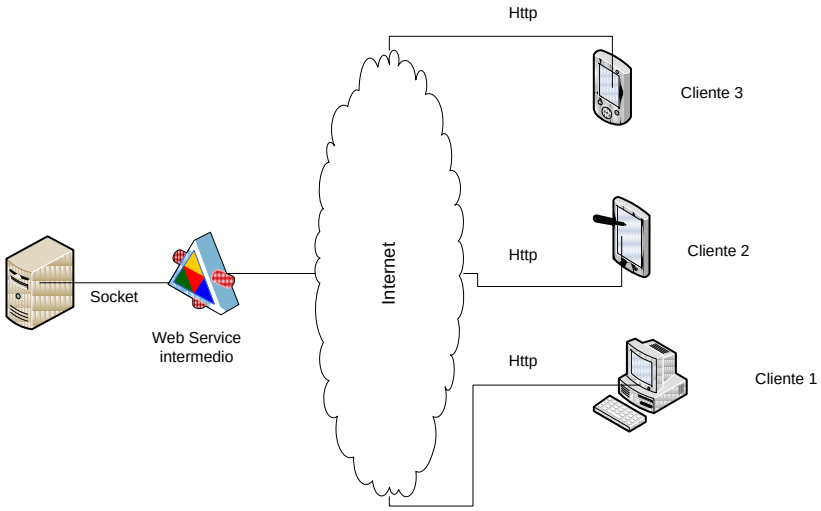
UML (*Unified Modeling Lenguaje* o Lenguaje Unificado de Modelado) es un lenguaje de modelado de sistemas software, aunque todavía no es un estándar, está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema software.

4.1. Arquitectura del sistema desde el punto de vista de las comunicaciones

En esta sección se presentan distintas alternativas a elegir a la hora de implantar una arquitectura de comunicaciones entre distintos clientes y un servicio que muestre como interfaz una aplicación 3D implementada con XNA

4.1.1. Arquitectura 1: Web-Service Intermedio entre clientes y servidor

4.1.1.1. Arquitectura 1.A: uso de sockets entre el Web-Service y el servidor



Ventajas

- Sencilla implementación entre cliente y Web-Service
- Clientes multiplataforma (Linux, java, smartclients o thinclients con interfaces adaptadas)
- Evadir firewalls ya que emplea puerto 80.

Desventajas

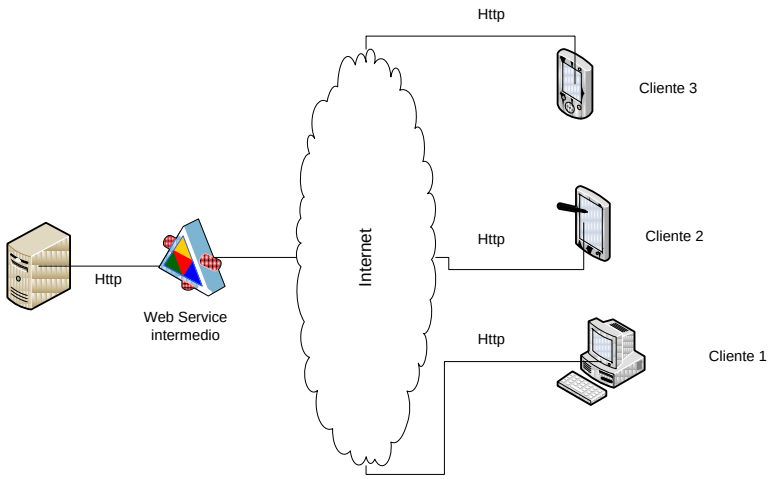
- Complejidad en la implementación del socket, ya que se requiere que el servidor cree hilos para atender múltiples peticiones desde el Web-Service.
- Requiere implementar un Web-Service y emplear un servidor web IIS.

Descripción del funcionamiento

Se deberá desplegar un Web-Service que será el responsable de capturar las órdenes de cada cliente. Mientras tanto en el servidor (o posiblemente en cualquier otra máquina) se tendrá al juego escuchando un socket por el cual recibirá órdenes por parte del Web-Service.

Por otra parte los clientes consumirán el Web-Service indicando que acción debe realizar el avatar que los representa. Por cada cliente, cuando inician su ejecución se debe crear un avatar diferente, hasta un máximo de 6.

4.1.1.2. Arquitectura 1.B: uso de HTTP (soap) entre el Web-Service y el servidor



Ventajas

- Sencilla implementación entre cliente y Web-Service
- Clientes multiplataforma (Linux, java, smartclients o thinclients con interfaces adaptadas)
- Evadir firewalls ya que emplea puerto 80.
- Más parecido a cómo trabaja el juego, cada vez que lee del web service si tiene que hacer algo es como si leyese el estado del teclado.

Desventajas

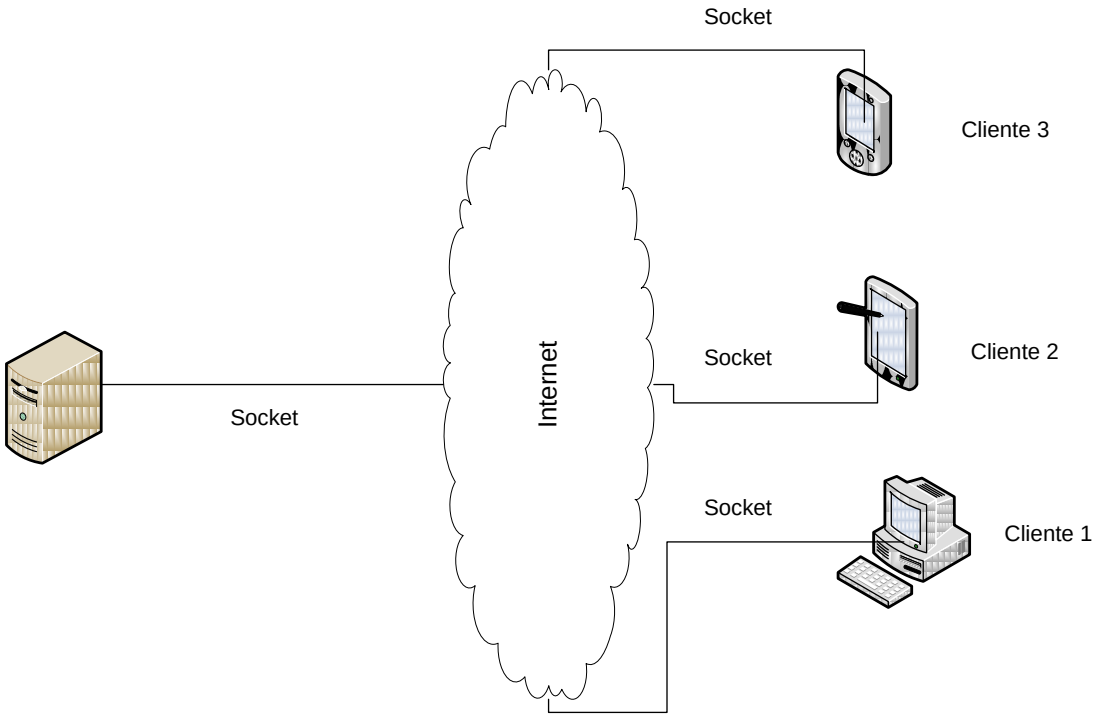
- Sobrecarga entre Servidor y Web-Service, ya que el servidor debe estar constantemente consumiendo el servicio web para conocer si debe realizar alguna acción (aunque la comunicación es en local, pero podría ser remota).
- Requiere implementar un Web-Service y emplear un servidor web IIS.

Breve descripción del funcionamiento

Se deberá desplegar un Web-Service que será el responsable de capturar las órdenes de cada cliente. Mientras tanto en el servidor (o posiblemente en cualquier otra máquina) se tendrá al juego lanzando consultas al Web-Service (consumiéndolo) cada vez que se llame al método Update (que es donde se suele actualizar todo el juego, como por ejemplo posiciones de los avatares, lecturas del teclado, calculo de físicas, etc…). Puesto que esto podría generar una carga excesiva en el Web-Service, se puede limitar para que se ejecute una vez cada decima de segundo, o según se vean las necesidades.

Por otra parte los clientes consumirán el Web-Service indicando que acción debe realizar el avatar que los representa. Por cada cliente, cuando inician su ejecución se debe crear un avatar diferente, hasta un máximo de6.

4.1.2. Arquitectura 2: Comunicación mediante Sockets



Ventajas

- Comunicación directa entre clientes y servidor.
- Sencilla conversión a trabajo con ficheros en modo local.
- Forma más común de hacerlo.

Desventajas

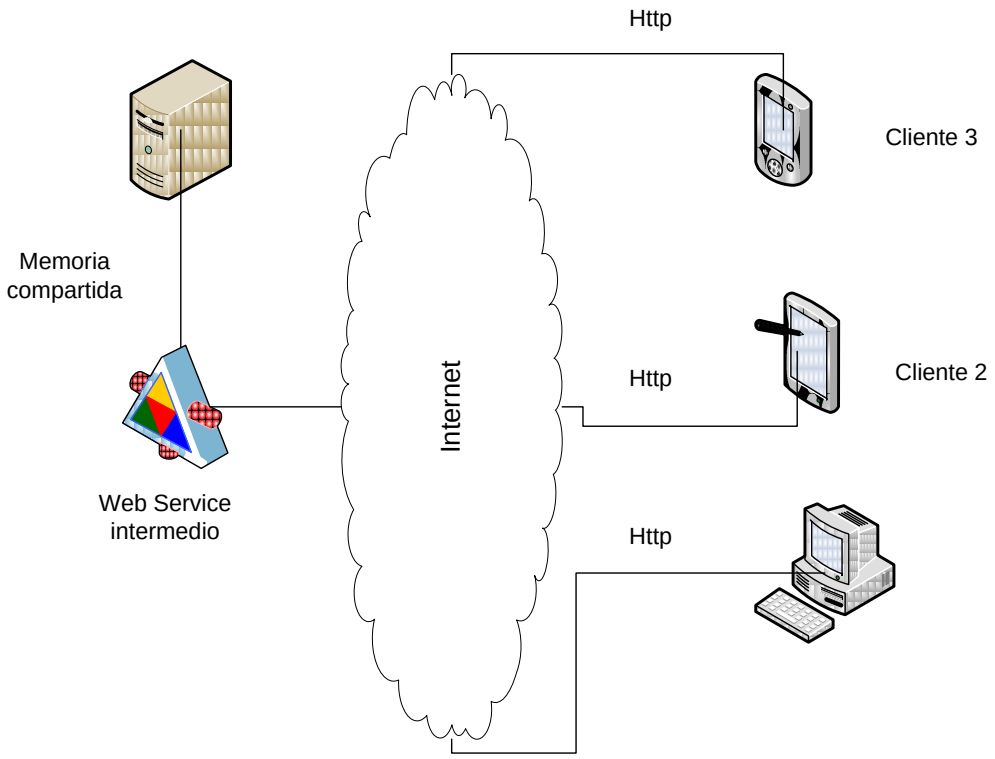
- Mayor complejidad de programación.
- Multithread para atender múltiples clientes.

Breve descripción del funcionamiento

Por un lado estará el servidor, que será el juego. Abrirá un socket y permanecerá a la escucha de que un cliente pida una conexión. Cuando un cliente conecte deberá crear un thread nuevo para atender dicha conexión y poder mantenerse a la escucha de nuevos clientes, hasta 6. Por cada cliente que se conecte se deberá crear un avatar diferente.

Los clientes crearán una conexión contra el servidor a través de un socket y podrán enviar movimientos u órdenes al servidor.

4.1.3. Arquitectura 3: Web-Service invocando al juego.



Ventajas

- Mismas que Arquitectura 1.B

Desventajas

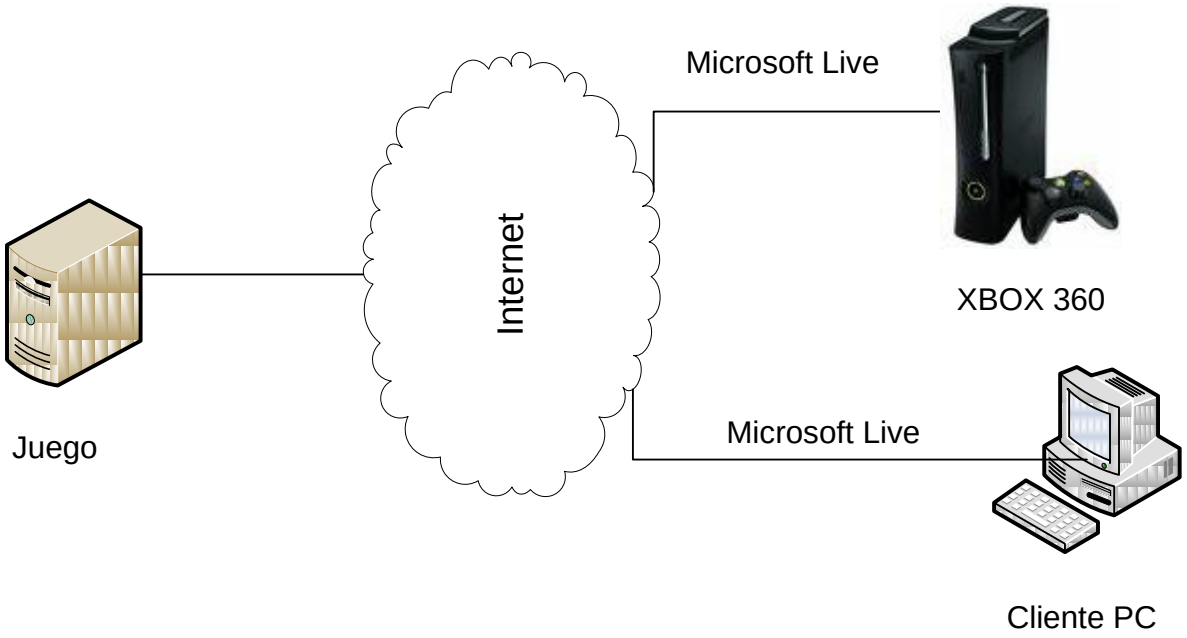
- Requiere un thread que invoque el juego desde el Web-Service, no sabemos si esto se puede hacer directamente. Pero desde una aplicación Windows forms se puede, generando una librería del juego y posteriormente lanzando un thread que cree la instancia del juego.

Breve descripción del funcionamiento

En esta arquitectura, cuando se invoca por primera vez el web service, éste instanciará el juego y lo lanzará, mostrándose la interfaz con un único avatar. Aun no se sabe si se puede hacer de esta manera, ya que el web-service reside en un servidor web y no se ha comprobado que se puedan lanzar aplicaciones con interfaz gráfica desde un web-service.

Los clientes actuarán de la misma manera que en la arquitectura 1.B.

4.1.4. Arquitectura 4: Plataforma Microsoft Live



Ventajas

- Compatible con XBOX 360.
- Aporta mecanismos de simulación de redes, para poder simular paquetes perdidos, etc…
- Según lo visto en libros parece sencillo.
- Muy enfocado a arquitecturas para videojuegos.

Desventajas

- Diferente para red Lan e internet.
- Para internet requiere una licencia de desarrollo.
- Probablemente los clientes deban ser XNA.

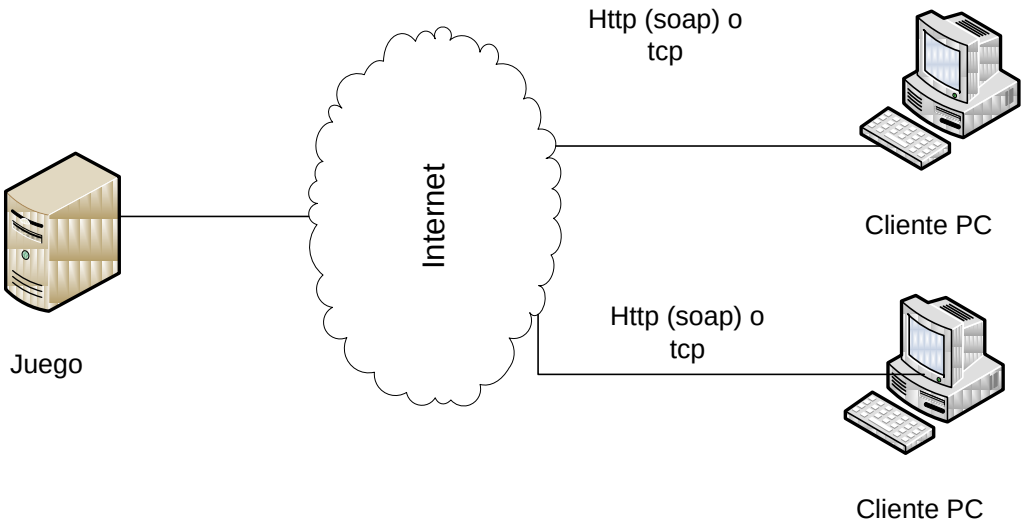
Breve descripción del funcionamiento

Esta es la arquitectura típica de un juego en red para Xbox o Pc sobre la plataforma Live.

El servidor registrará un componente para las comunicaciones, permaneciendo a la escucha de los clientes.

Los clientes también deberán registrar un componente para comunicaciones con el objetivo de poder enviar las órdenes al servidor.

4.1.5. Arquitectura 5: Empleo de .Net Remoting



Ventajas

- Implementación sencilla, similar a RMI.
- Permite comunicaciones sobre HTTP (soap) o directamente sobre TCP

Desventajas

- Requiere la creación de los distintos objetos remotos.
- Los clientes debe ser programados en .Net u otro lenguaje que soporte .Net Remoting.

Breve descripción del funcionamiento

El uso de .Net Remoting requiere la creación de una serie de objetos que serán los que estarán distribuidos. El servidor registrará dichos objetos para que los clientes puedan instanciarlos remotamente.

Este objeto remoto será el que contenga el juego, que en el servidor será lanzado, y en los clientes simplemente se empleará para enviar órdenes al juego remoto.

4.1.6. Elección final sobre las arquitecturas planteadas

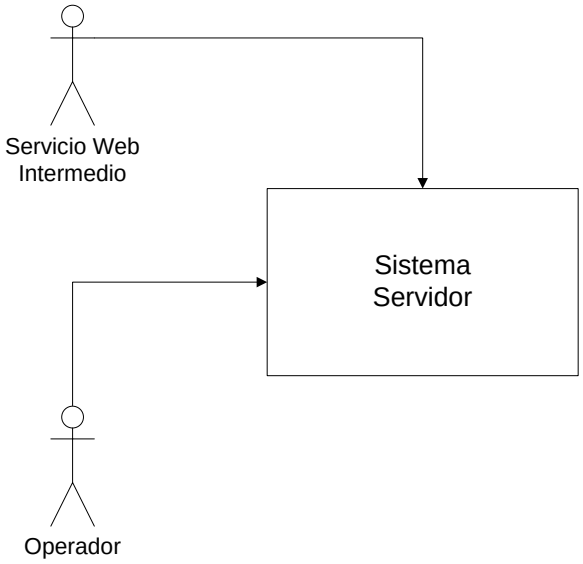
Después de realizar un estudio analizando ventajas y desventajas, así como la simplicidad se ha decidido implantar la arquitectura señalada en el punto 4.1.1.1.

Para la elección de esta arquitectura se ha pensado en la capacidad de concurrencia que ofrecen los web services, abstrayendo toda la complejidad y delegándola en el propio servidor web. Esto posibilita que el lado cliente se haga más escalable. Así mismo, se decidió por emplear sockets entre el web service y el servidor de la aplicación. Esto permite que el servidor web este en una máquina y que el motor de XNA este en otra, permitiendo tener una máquina dedicada solo para las tareas más pesadas como son las de render de modelos.

4.2.Aplicación para el servidor

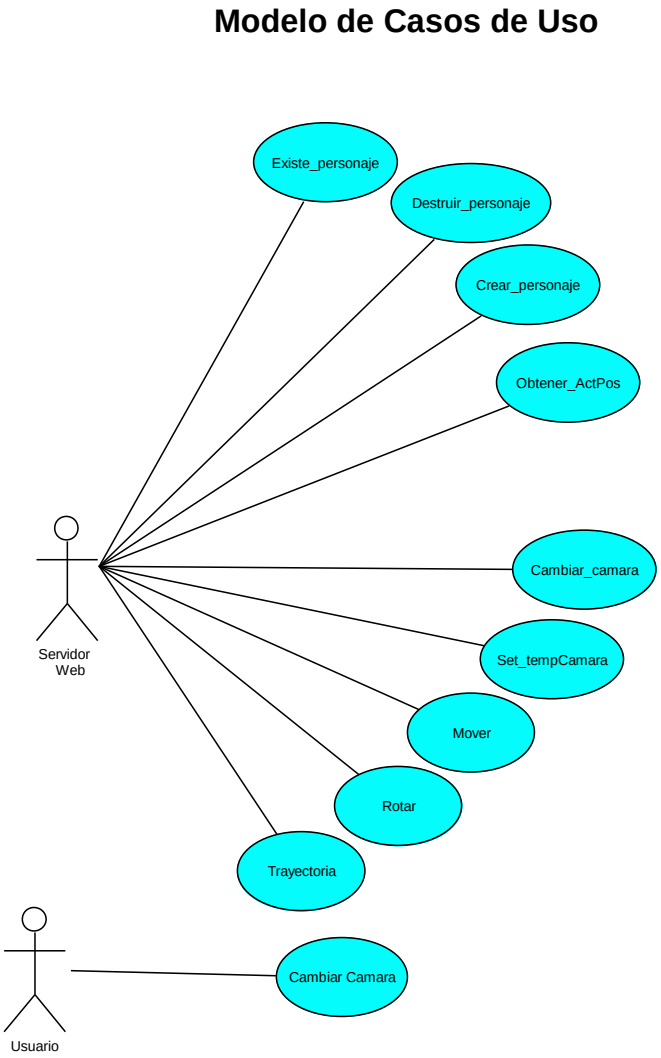
Con el objetivo de ilustrar el proceso de análisis de la aplicación destinada al servidor, se han empleado diagramas de casos de uso, mostrando que hace cada tipo de usuario, diagramas de actividad y secuencia que ilustran el proceso que sigue cada uno de los casos de uso y finalmente un modelo de clases de diseño, que dará una visión general de las clases que se deberán emplear en la fase de desarrollo.

4.2.1. Diagrama de contexto del servidor



El presente diagrama de contexto muestra una primera visión de alto nivel del servidor. En este diagrama se ven representados los dos posibles usuarios que interactúan con el sistema, por un lado tendremos un operador, que será la persona que estará delante de la máquina servidora, por otro lado está el servicio web intermedio, que iniciará las acciones necesarias en el servidor.

4.2.2. Casos de uso del servidor.



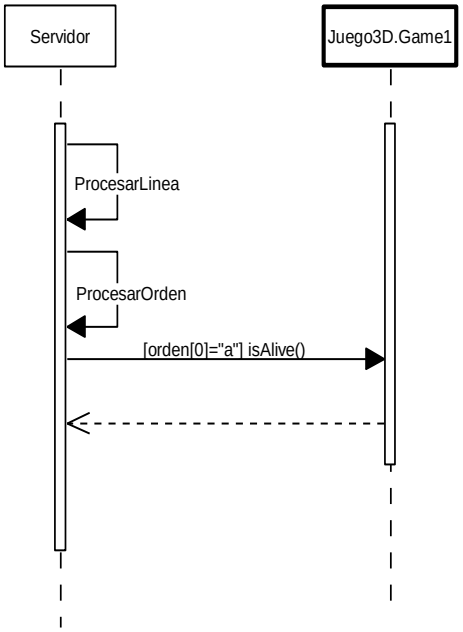
4.2.3. Especificación de los casos de uso y diagramas de actividad

A continuación se mostrarán los distintos diagramas de actividad correspondientes a cada caso de uso del servidor, junto con una descripción de cada uno de los casos de uso.

4.2.3.1. Existe personaje

Nombre	Existe personaje
Actores	Servidor web
Descripción	Este caso de uso permite al sistema conocer si existe un personaje en el sistema con el nombre que se proporciona como parámetro.

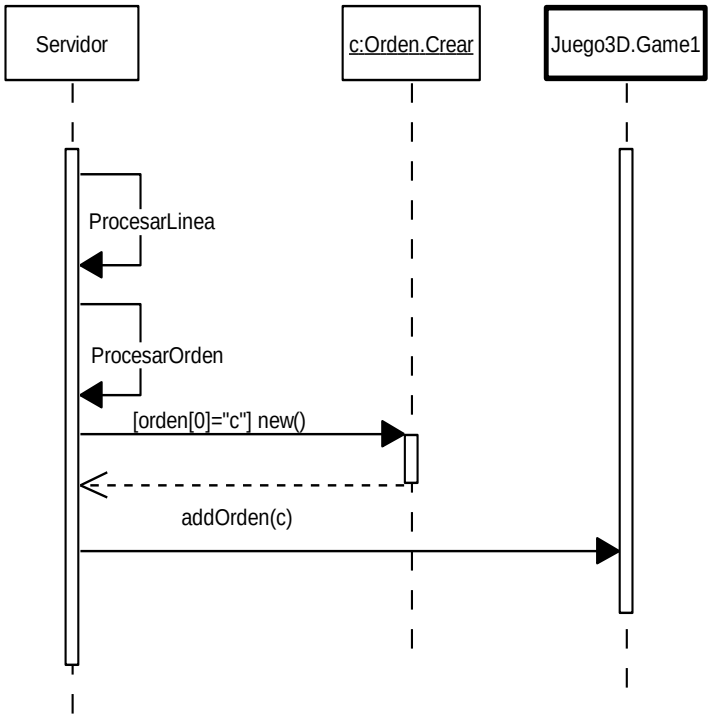
EXISTE PERSONAJE



4.2.3.2. Crear personaje

Nombre	Crear personaje
Actores	Servidor web
Descripción	Este caso de uso permite al sistema crear un personaje con el nombre que se le indica y empleando el modelo de personaje indicado. Si el personaje ya existe no se hace nada.

CREAR PERSONAJE

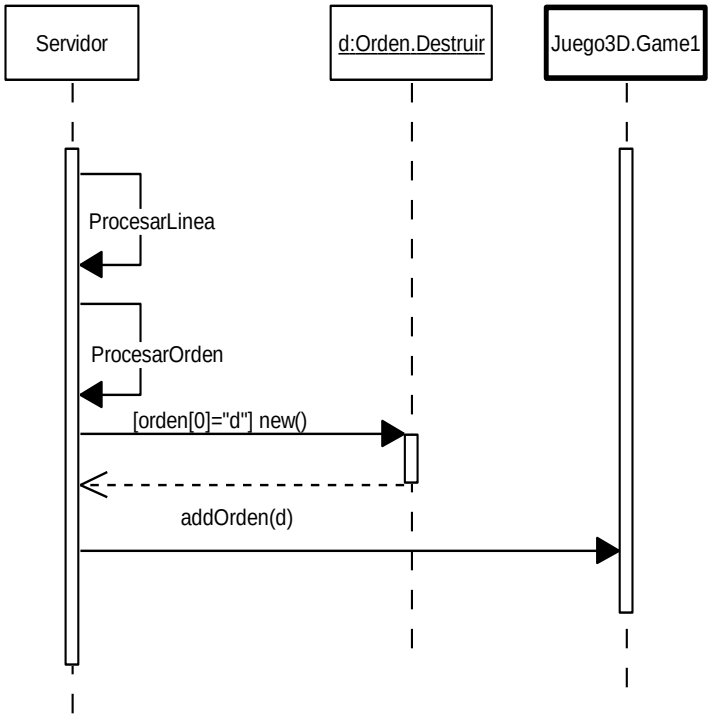


4.2.3.3. Destruir personaje

Nombre	Crear personaje
Actores	Servidor web
Descripción	Este caso de uso permite al sistema destruir un personaje con el nombre que

se le indica, si no existe el personaje no se hace nada.

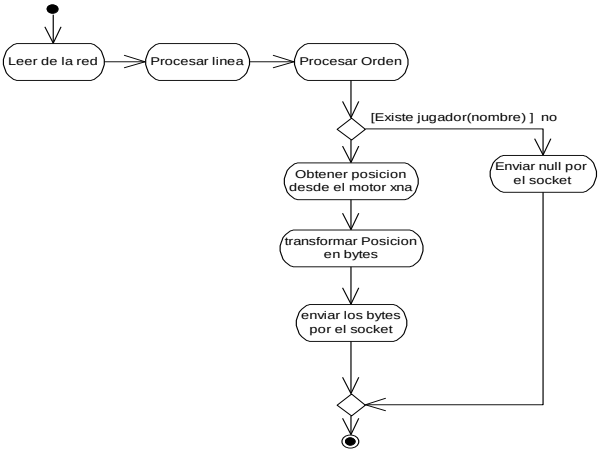
DESTRUIR PERSONAJE



4.2.3.4. Obtener actPos

Nombre	Obtener actPos
Actores	Servidor web
Descripción	Este caso de uso permite al sistema devolver la posición actual en la que se encuentra el personaje cuyo nombre se especifica en la orden.

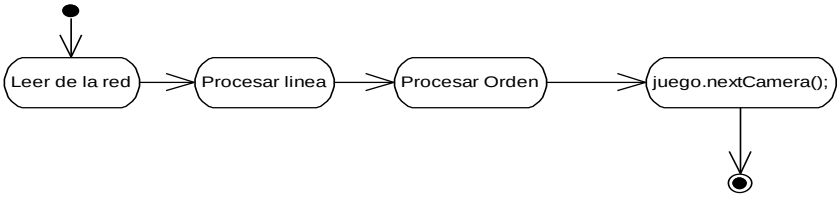
Obtener actPos



4.2.3.5. Cambiar cámara ws

Nombre	Cambiar cámara ws
Actores	Servidor web
Descripción	Este caso de uso permite al sistema recibir una orden para cambiar la cámara que está en uso. Las cámaras están numeradas por lo se cambiará a la siguiente cámara en el orden especificado.

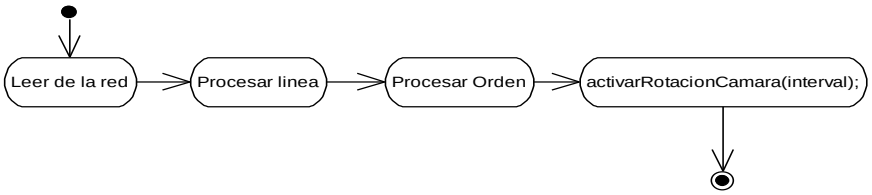
Cambiar cámara ws



4.2.3.6. Set tempCamara

Nombre	Set tempCamara
Actores	Servidor web
Descripción	Este caso de uso permite al sistema recibir una orden para cambiar el intervalo de tiempo entre un cambio de cámara y el siguiente. Cuando se recibe por primera vez se activa el temporizador. Si el valor indicado es cero, se detiene el temporizador

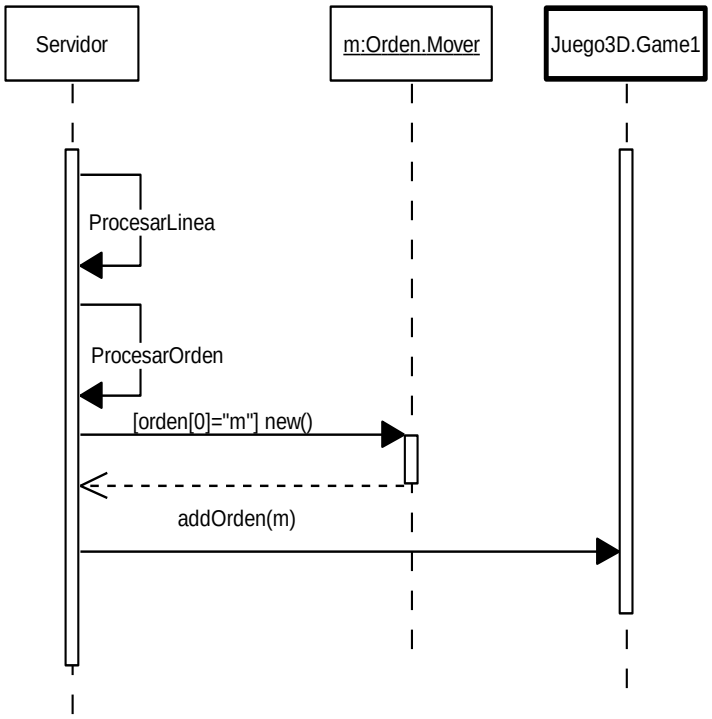
Set tempCamara



4.2.3.7. Mover

Nombre	Mover
Actores	Servidor web
Descripción	Este caso de uso permite al sistema recibir una orden para mover a un personaje indicado por el nombre de personaje, la cantidad de movimiento en cada coordenada especificada en la orden.

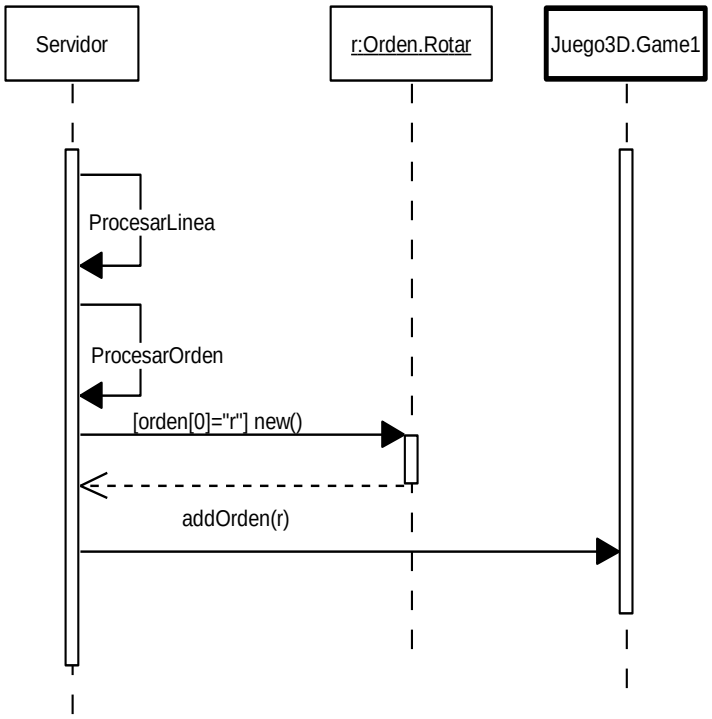
MOVER PERSONAJE



4.2.3.8. Rotar

Nombre	Rotar
Actores	Servidor web
Descripción	Este caso de uso permite al sistema recibir una orden para rotar a un personaje indicado por el nombre de personaje, la cantidad de rotación va indicada en la orden. La dirección de giro la indica el signo de la cantidad.

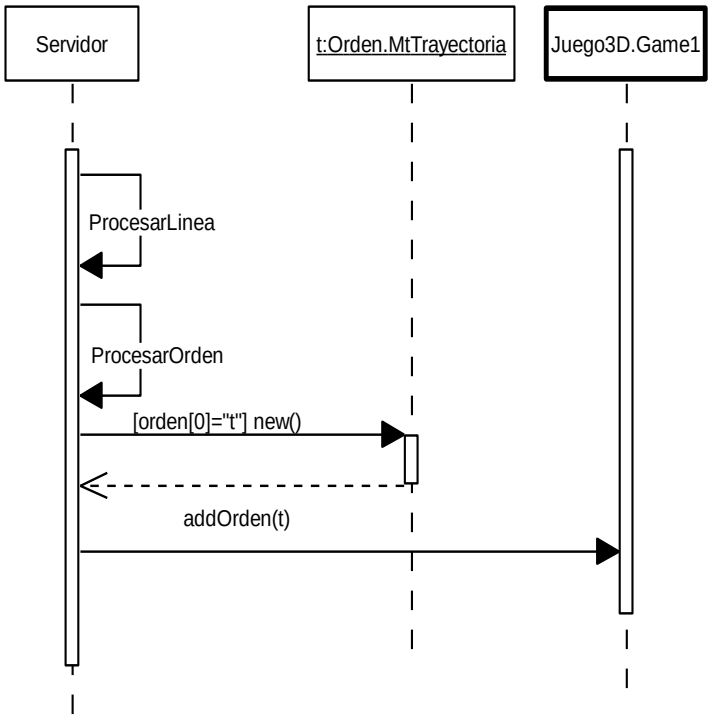
ROTAR PERSONAJE



4.2.3.9. Trayectoria

Nombre	Trayectoria
Actores	Servidor web
Descripción	Este caso de uso permite al sistema recibir una orden para mover el personaje, indicado por el nombre de personaje, a la posición que se indica en la orden.

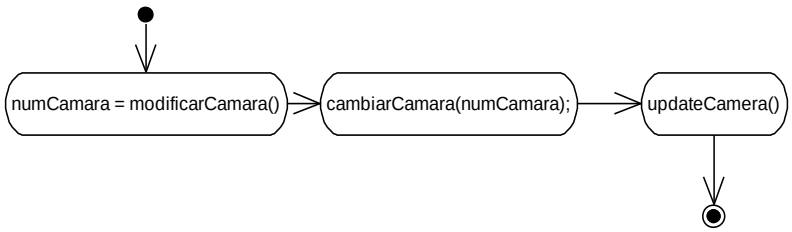
TRAYECTORIA



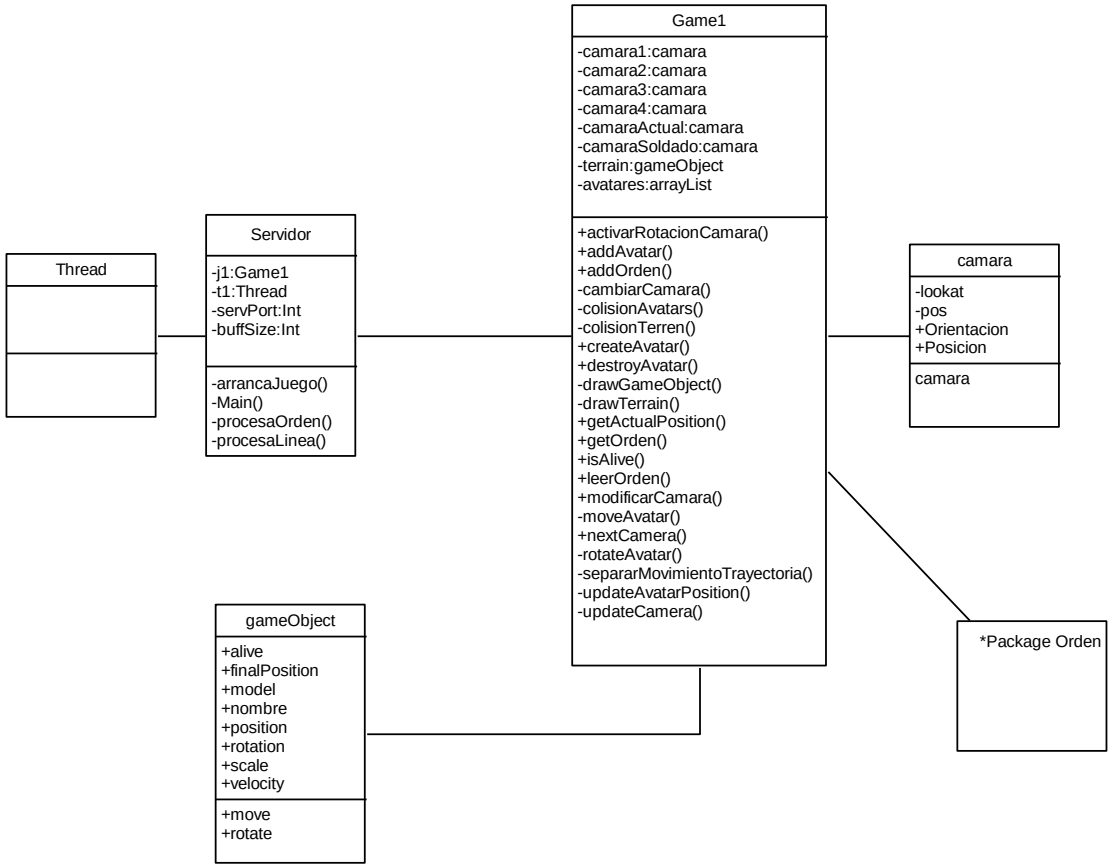
4.2.3.10. Cambiar cámara

Nombre	Cambiar cámara
Actores	Usuario
Descripción	Este caso de uso permite que un usuario le indique al sistema de render de xna modificar la cámara actual por otra de las 6 que existen en el sistema. Para ello se emplearán los números del 1 al 6 del teclado numérico

Cambiar cámara



4.2.4. Modelo de clases del servidor

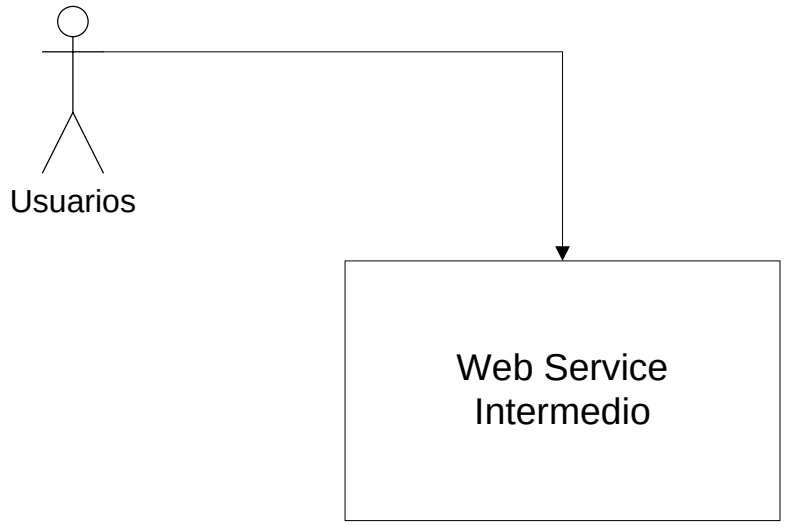


El contenido del paquete Orden se detalla en el punto 4.6.1 debido a su importancia dentro del sistema

4.3.Web Service

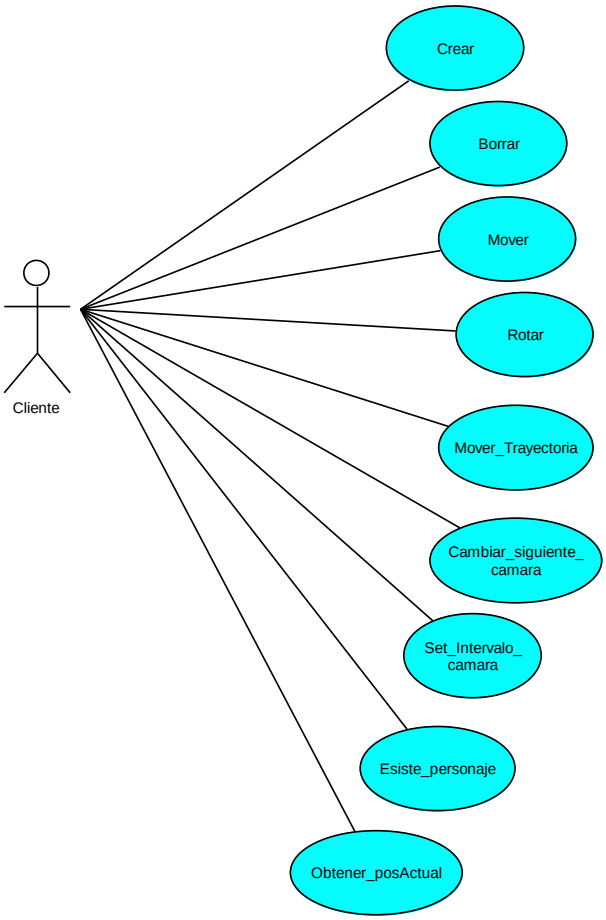
Con el objetivo de ilustrar el proceso de análisis del web service, se han empleado diagramas de casos de uso, mostrando que hace cada tipo de usuario, diagramas de actividad y secuencia que ilustran el proceso que sigue cada uno de los casos de uso y finalmente un modelo de clases de diseño, que dará una visión general de las clases que se deberán emplear en la fase de desarrollo.

4.3.1. Diagrama de contexto del servidor



El presente diagrama de contexto muestra una primera visión de alto nivel del web service. En este diagrama se ve representado el único tipo de usuario que interactúa con el sistema.

4.3.2. Casos de uso del Web Service.



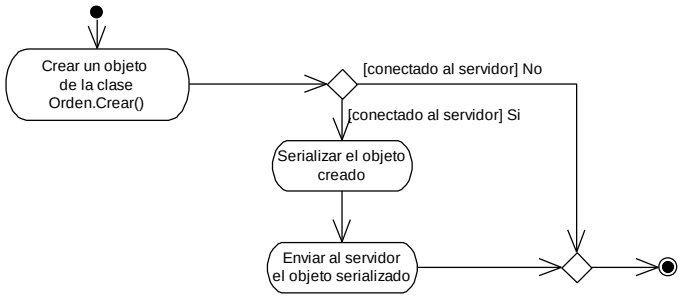
4.3.3. Especificación de los casos de uso del Web Service

A continuación se mostrarán los distintos diagramas de actividad correspondientes a cada caso de uso del Web Service, junto con una descripción de cada uno de los casos de uso.

4.3.3.1. Crear

Nombre	Crear
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.Crear para realizar la creación de un personaje, serializarlo y enviarlo al servidor de render.

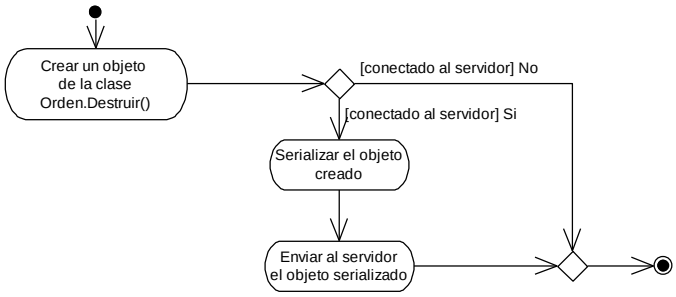
Crear



4.3.3.2. Borrar

Nombre	Borrar
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.Distruir para realizar el borrado de un personaje, serializarlo y enviarlo al servidor de render.

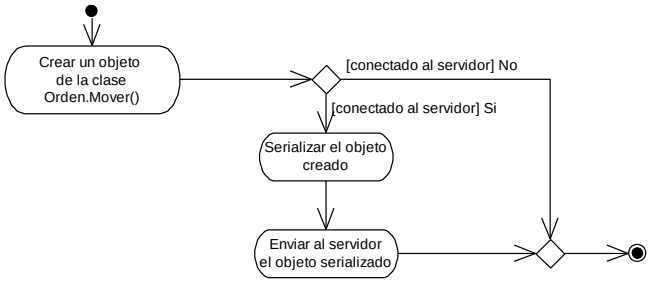
Borrar



4.3.3.3. Mover

Nombre	Mover
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.Mover para realizar el movimiento de un personaje, serializarlo y enviarlo al servidor de render.

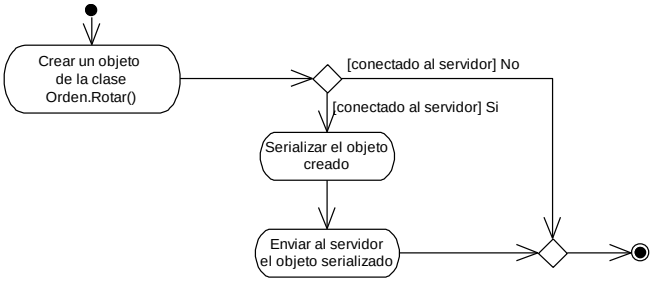
Mover



4.3.3.4. Rotar

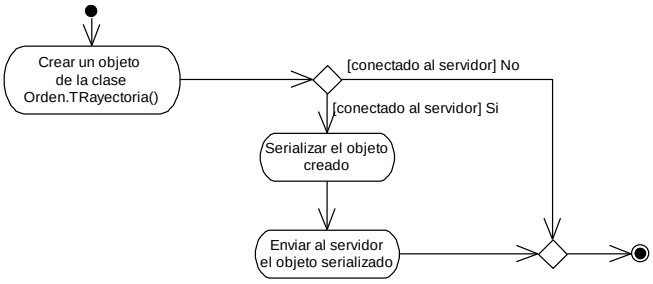
Nombre	Mover
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.Rotar para realizar la rotación de un personaje, serializarlo y enviarlo al servidor de render.

Rotar



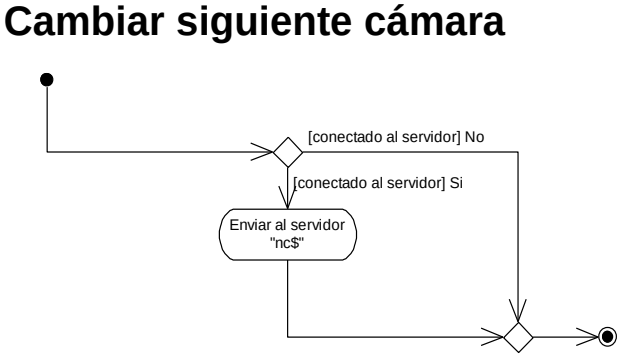
4.3.3.5. Mover trayectoria

Nombre	Mover trayectoria
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.MTrayectoria para realizar el movimiento de un punto inicial a un punto final de un personaje, serializarlo y enviarlo al servidor de render.



4.3.3.6. Cambiar siguiente cámara

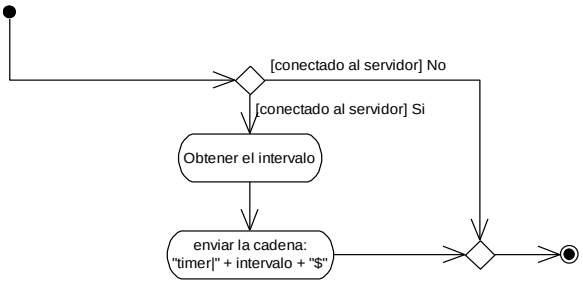
Nombre	Cambiar siguiente cámara
Actores	Cliente
Descripción	Este caso de uso se encarga enviar al servidor de render la cadena “nc$” al servidor para realizar el cambio a la siguiente cámara.



4.3.3.7. *Set intervalo cámara*

Nombre	Set intervalo cámara
Actores	Cliente
Descripción	Este caso de uso se encarga enviar al servidor de render una orden para activar un temporizador que cambie de cámara automáticamente. El parámetro de tiempo lo obtiene como argumento del método remoto. Para ello crea la cadena “timer|” + intervalo + “$”.

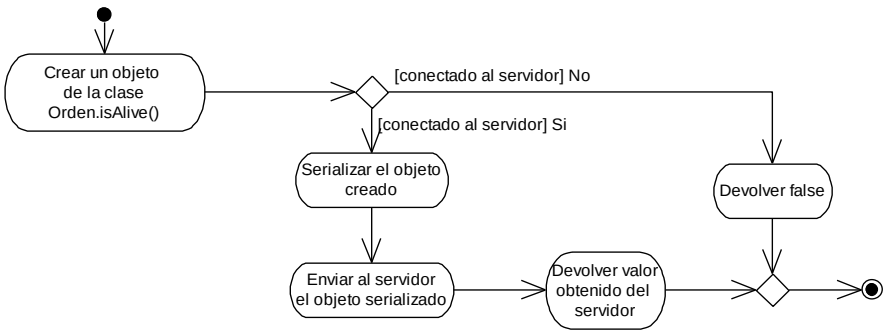
Set intervalo cámara



4.3.3.8. *Existe personaje*

Nombre	Existe personaje
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.isAlive para comprobar la existencia en el sistema de un personaje con un nombre dado.

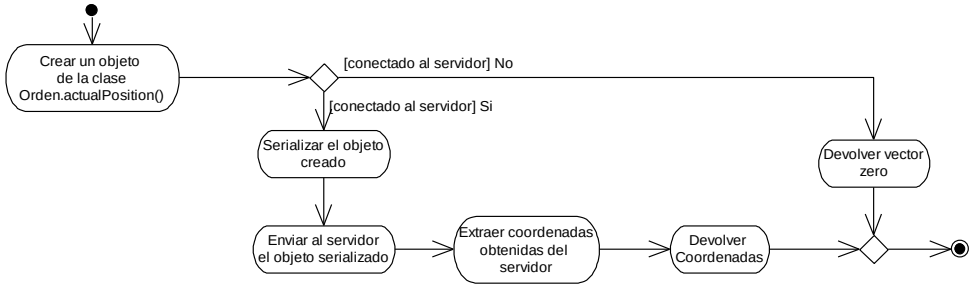
Existe personaje



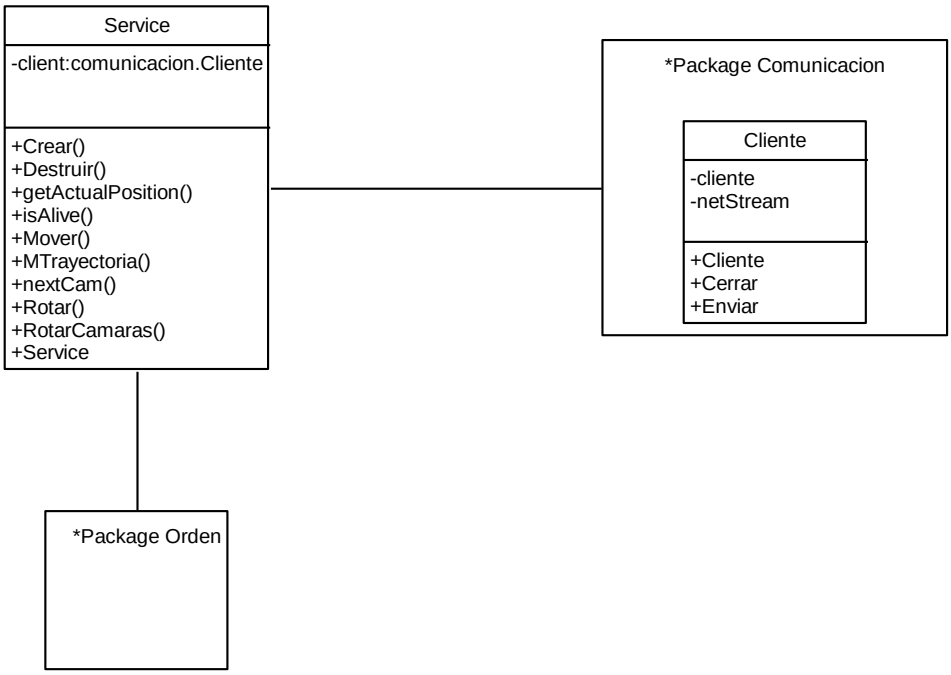
4.3.3.9. Obtener posActual

Nombre	Obtener posActual
Actores	Cliente
Descripción	Este caso de uso se encarga de crear un objeto de la clase Orden.actualPosition obtener la posición que ocupa un personaje indicado por su nombre.

Obtener posActual



4.3.4. Modelo de clases del Web Service

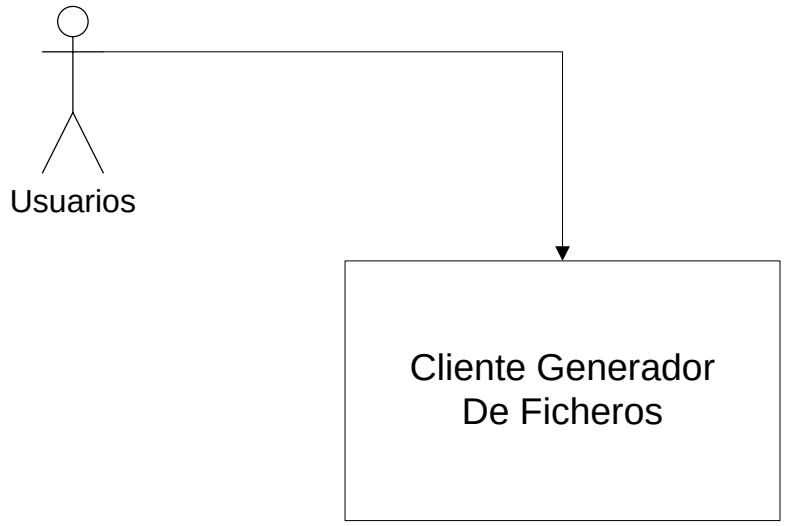


*El contenido del paquete Orden se detalla en el punto 4.6.1 debido a su importancia dentro del sistema.

4.4.Cliente generador de ficheros

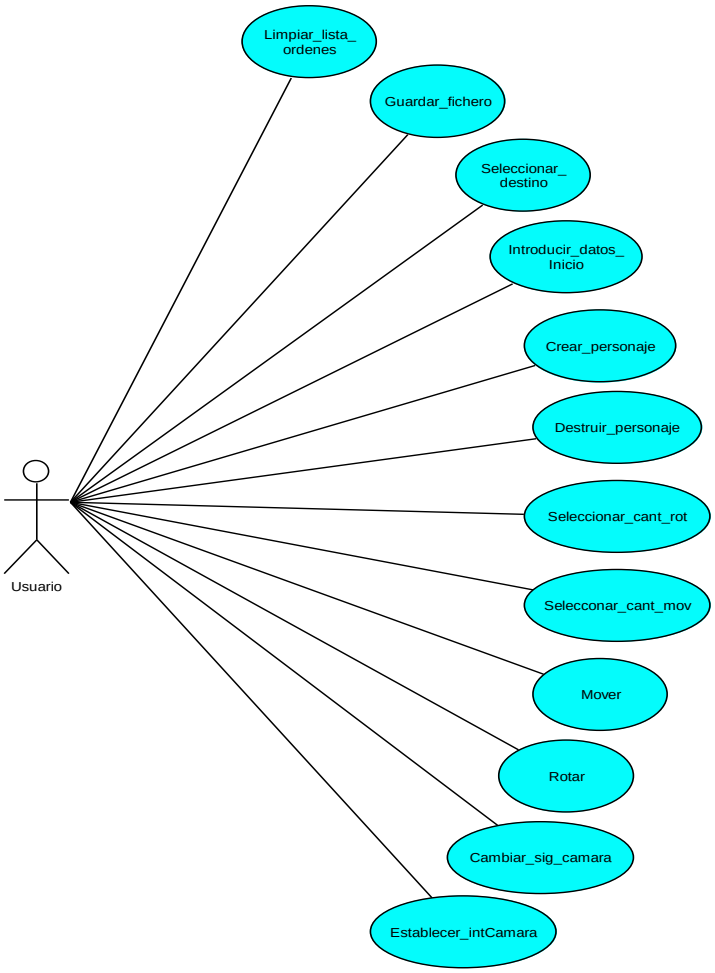
Con el objetivo de ilustrar el proceso de análisis del cliente generador de ficheros, se han empleado diagramas de casos de uso, mostrando que hace cada tipo de usuario, diagramas de actividad y secuencia que ilustran el proceso que sigue cada uno de los casos de uso y finalmente un modelo de clases de diseño, que dará una visión general de las clases que se deberán emplear en la fase de desarrollo.

4.4.1. Diagrama de contexto del servidor



El presente diagrama de contexto muestra una primera visión de alto nivel del cliente generador de ficheros. En este diagrama se ve representado el único tipo de usuario que interactúa con el sistema.

4.4.2. Casos de uso del cliente Generador de ficheros.

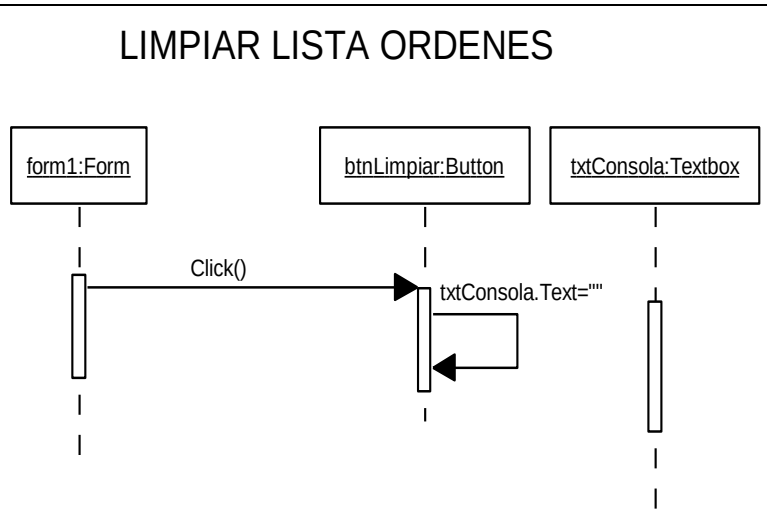


4.4.3. Especificación de los casos de uso del cliente generador de ficheros

A continuación se mostrarán los distintos diagramas de actividad correspondientes a cada caso de uso del cliente generador de ficheros, junto con una descripción de cada uno de los casos de uso.

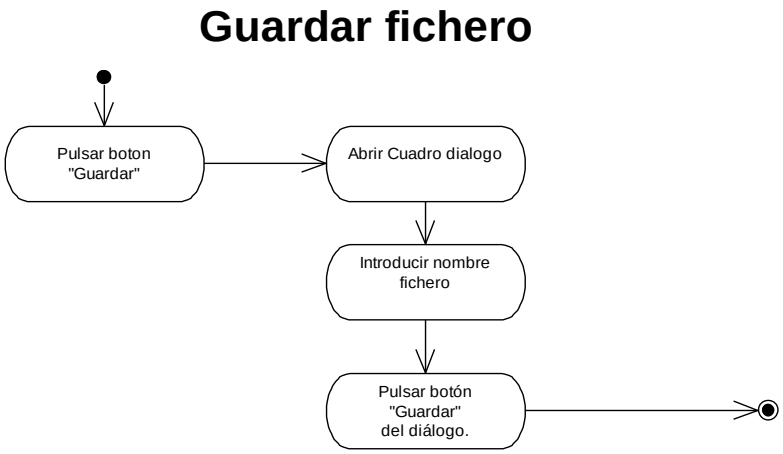
4.4.3.1. *Limpiar lista órdenes*

Nombre	Limpiar lista órdenes
Actores	Usuario
Descripción	Este caso de uso permite al usuario limpiar la lista que contiene las órdenes que se han ido realizado con el objetivo de iniciar una nueva o bien de evitar que se almacenen en un fichero.



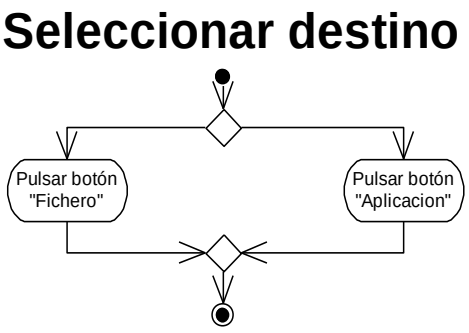
4.4.3.2. *Guardar fichero*

Nombre	Guardar fichero
Actores	Usuario
Descripción	Este caso de uso permite al usuario guardar en un fichero de texto el contenido de la lista de órdenes que el usuario ha generado.



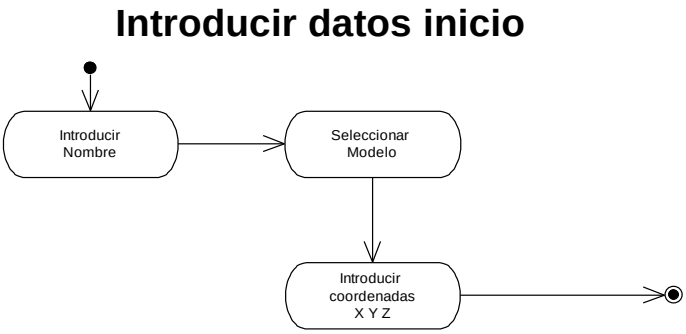
4.4.3.3. Seleccionar destino

Nombre	Seleccionar destino
Actores	Usuario
Descripción	Este caso de uso permite al usuario elegir de qué manera quiere que trabaje el sistema. Puede elegir que las órdenes se agreguen a una lista o bien mostrar una interfaz gráfica que muestre el entorno de render y ver las acciones que va realizando el personaje.



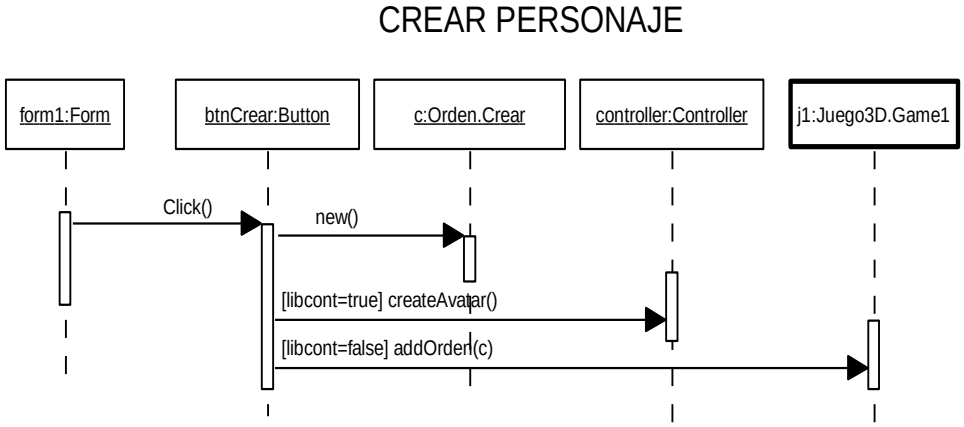
4.4.3.4. Introducir datos inicio

Nombre	Introducir datos inicio
Actores	Usuario
Descripción	Este caso de uso permite al usuario introducir el nombre, las coordenadas de inicio y seleccionar el modelo para un personaje.



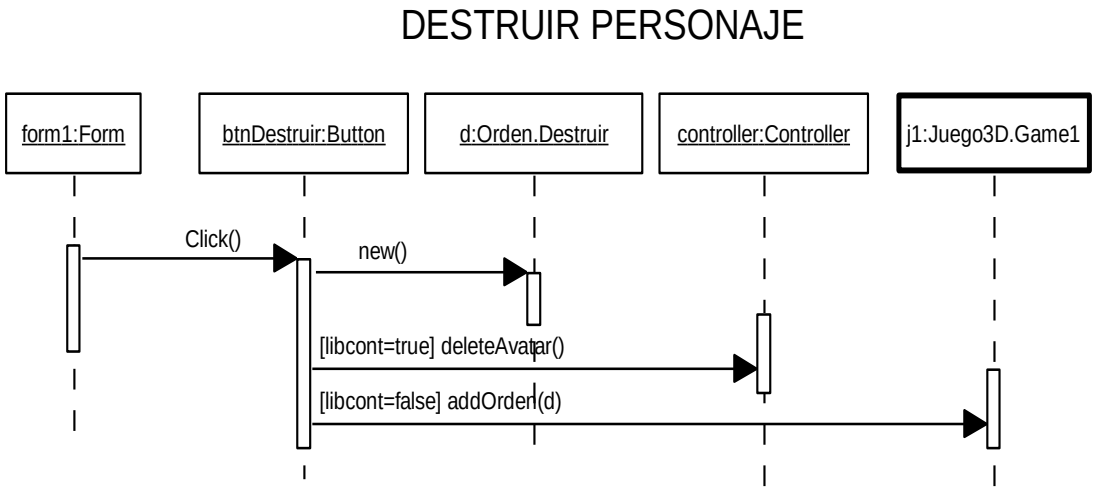
4.4.3.5. Crear personaje

Nombre	Crear personaje
Actores	Usuario
Descripción	Este caso de uso permite al usuario añadir una orden de creación de personaje a la lista de órdenes. Antes deberá haber realizado el caso de uso Introducir datos inicio.



4.4.3.6. *Destruir personaje*

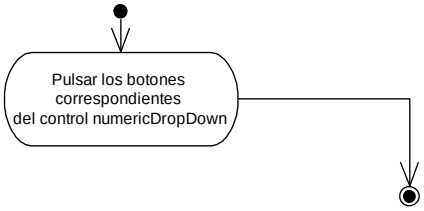
Nombre	Destruir personaje
Actores	Usuario
Descripción	Este caso de uso permite al usuario añadir una orden de destrucción de un personaje. Antes deberá haber realizado el caso de uso Crear personaje.



4.4.3.7. *Seleccionar cantidad rot*

Nombre	Seleccionar cantidad rot
Actores	Usuario
Descripción	Este caso de uso permite al usuario seleccionar una cantidad para realizar la rotación de un personaje. Para seleccionar el valor deseado el usuario deberá pulsar los botones correspondientes al control numericDropDown de rotación.

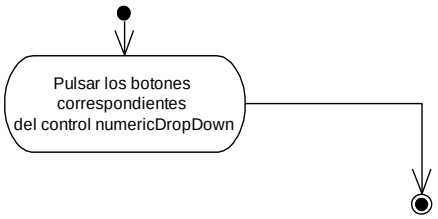
Seleccionar cantidad rot



4.4.3.8. Seleccionar cantidad mov

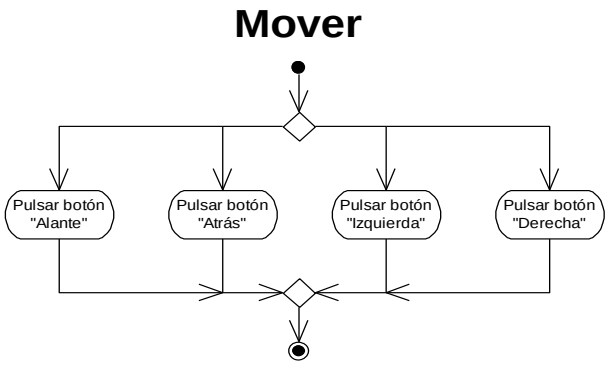
Nombre	Seleccionar cantidad mov
Actores	Usuario
Descripción	Este caso de uso permite al usuario seleccionar una cantidad para realizar el movimiento de un personaje. Para seleccionar el valor deseado el usuario deberá pulsar los botones correspondientes al control numericDropDown de movimiento.

Seleccionar cantidad Mov



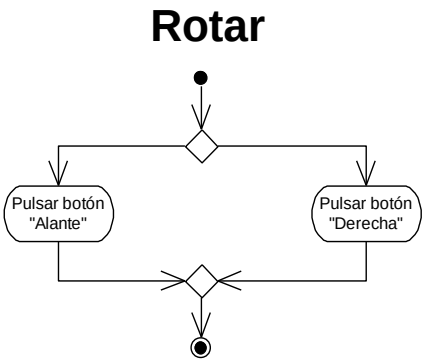
4.4.3.9. Mover

Nombre	Mover
Actores	Usuario
Descripción	Este caso de uso permite al usuario añadir una orden de movimiento a la lista de órdenes. Previamente habrá que haber realizado el caso de uso Crear personaje.



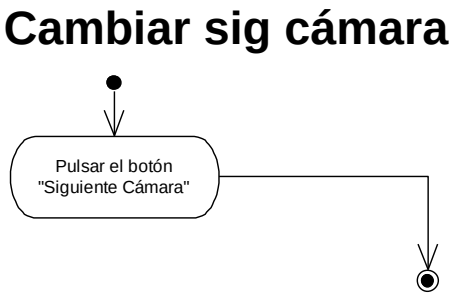
4.4.3.10. Rotar

Nombre	Rotar
Actores	Usuario
Descripción	Este caso de uso permite al usuario añadir una orden de rotación a la lista de órdenes. Previamente habrá que haber realizado el caso de uso Crear personaje.



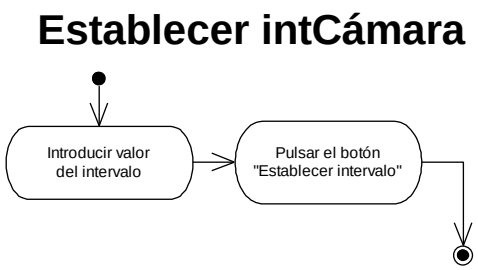
4.4.3.11. Cambiar sig cámara

Nombre	Cambiar sig cámara
Actores	Usuario
Descripción	Este caso de uso permite al usuario enviar una orden para cambiar a la siguiente cámara.



4.4.3.12. Establecer intCámara

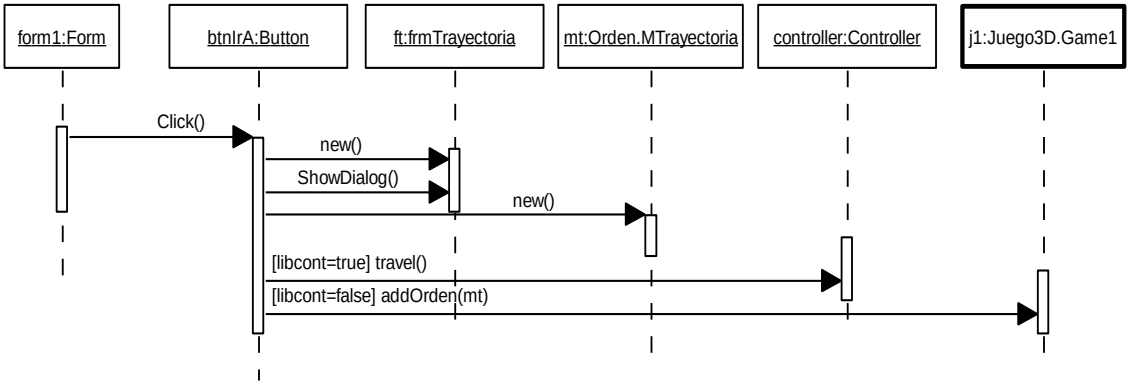
Nombre	Establecer intCámara
Actores	Usuario
Descripción	Este caso de uso permite al usuario enviar una orden para establecer un intervalo de tiempo para realizar el cambio de cámara de manera automática. Si el valor del intervalo es 0, el temporizador se detiene.



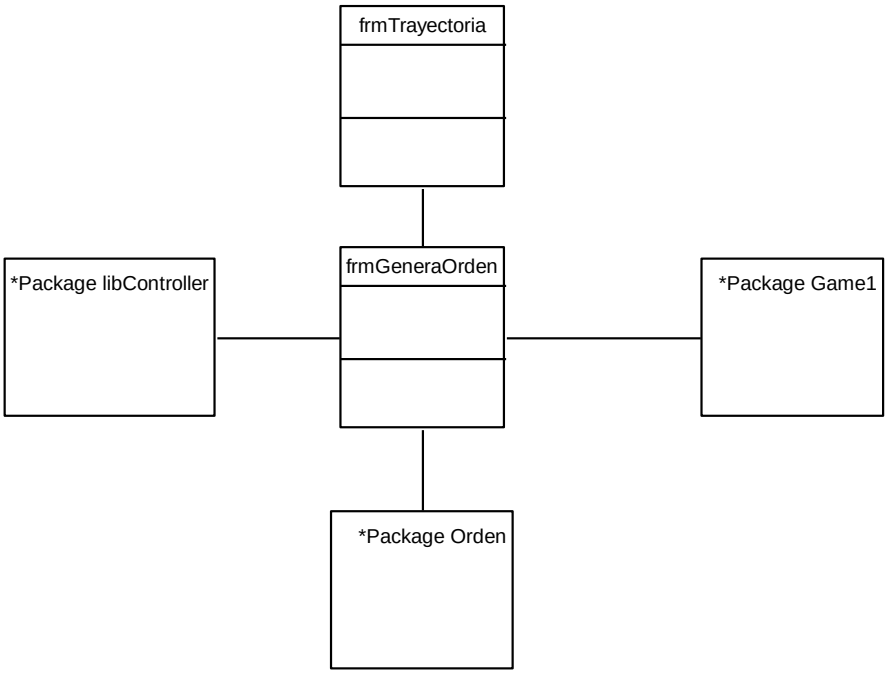
4.4.3.13. Mover por trayectoria

Nombre	Seleccionar destino
Actores	Usuario
Descripción	Este caso de uso permite al usuario introducir las coordenadas de destino para un personaje y generar la orden de movimiento por trayectoria hasta esa posición.

SELECCIONAR DESTINO



4.4.4. Modelo de clases del cliente generador de ficheros



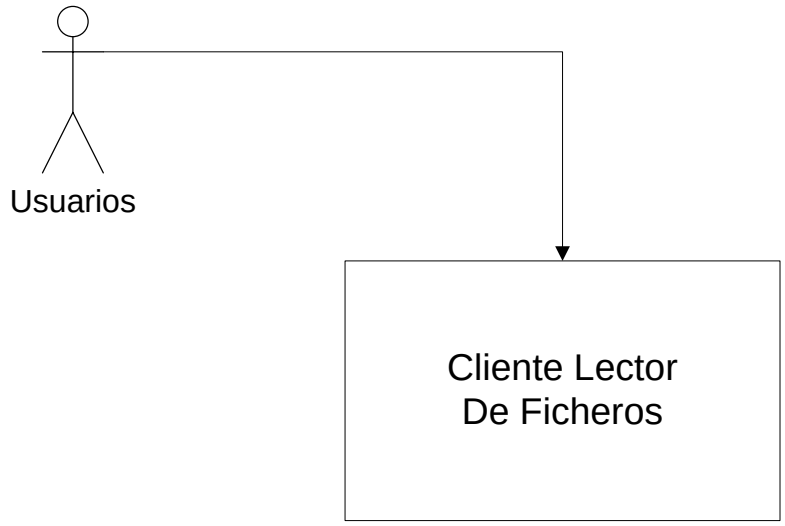
*Nota: los métodos y propiedades de los formularios no se han detallado, ya que responden a los controles que contendrán y no se detallan en fase de análisis.

*Nota2: el paquete Game1 está detallado en el punto 4.2.3. Los paquetes Orden y libController se detallarán en los puntos 4.6.1 Y 4.6.2 respectivamente.

4.5.Cliente lector de ficheros

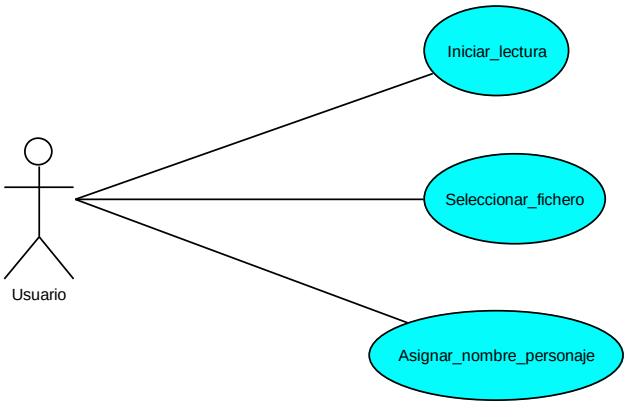
Con el objetivo de ilustrar el proceso de análisis del cliente lector de ficheros, se han empleado diagramas de casos de uso, mostrando que hace cada tipo de usuario, diagramas de actividad y secuencia que ilustran el proceso que sigue cada uno de los casos de uso y finalmente un modelo de clases de diseño, que dará una visión general de las clases que se deberán emplear en la fase de desarrollo.

4.5.1. Diagrama de contexto del servidor



El presente diagrama de contexto muestra una primera visión de alto nivel del cliente lector de ficheros. En este diagrama se ve representado el único tipo de usuario que interactúa con el sistema.

4.5.2. Casos de uso del cliente lector de ficheros.

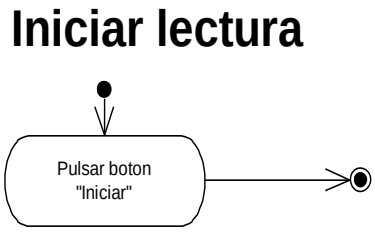


4.5.3. Especificación de los casos de uso del cliente lector de ficheros

A continuación se mostrarán los distintos diagramas de actividad correspondientes a cada caso de uso del cliente lector de ficheros, junto con una descripción de cada uno de los casos de uso.

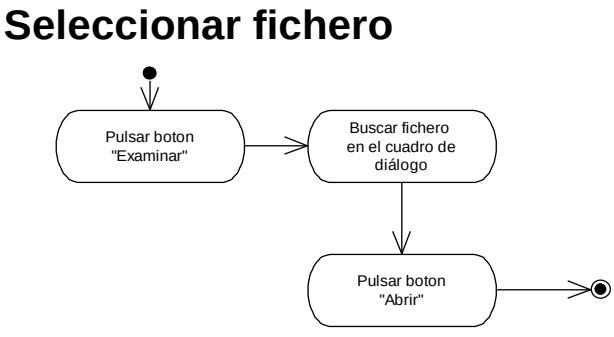
4.5.3.1. Iniciar lectura

Nombre	Iniciar lectura
Actores	Usuario
Descripción	Este caso de uso permite al usuario iniciar la lectura de un fichero de texto que contenga las órdenes que se desean enviar al servidor. Antes habrá que haber ejecutado los casos de uso Seleccionar fichero y Asignar nombre personaje.



4.5.3.2. Seleccionar fichero

Nombre	Seleccionar fichero
Actores	Usuario
Descripción	Este caso de uso permite al usuario buscar un fichero de texto existente en la máquina local y seleccionarlo para su posterior lectura.

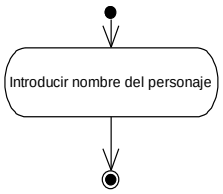


4.5.3.3. Asignar nombre personaje

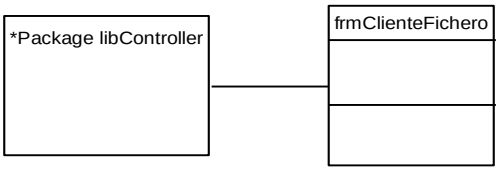
Nombre	Asignar nombre personaje

Actores	Usuario
Descripción	Este caso de uso permite al usuario introducir el nombre que se le dará al personaje correspondiente al usuario

Asignar nombre personaje



4.5.4. Modelo de clases del cliente lector de ficheros



Nota: los métodos y propiedades de los formularios no se han detallado, ya que responden a los controles que contendrán y no se detallan en fase de análisis.

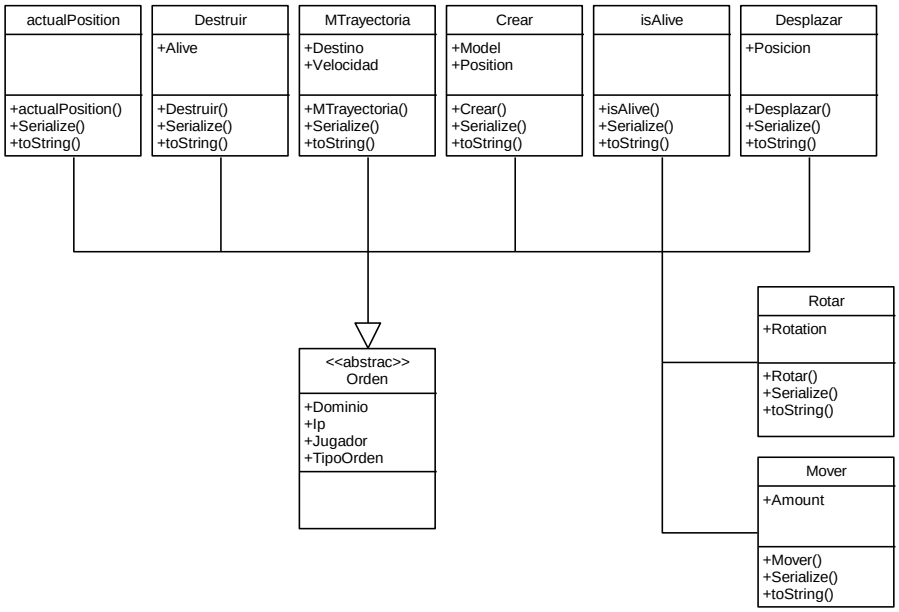
*Nota2: El paquete libController se detallará en el punto 4.6.2.

4.6.Modelos de clases de los paquetes libController y Orden

En este apartado se detallan los modelos de análisis de los dos paquetes que se emplean en el sistema.

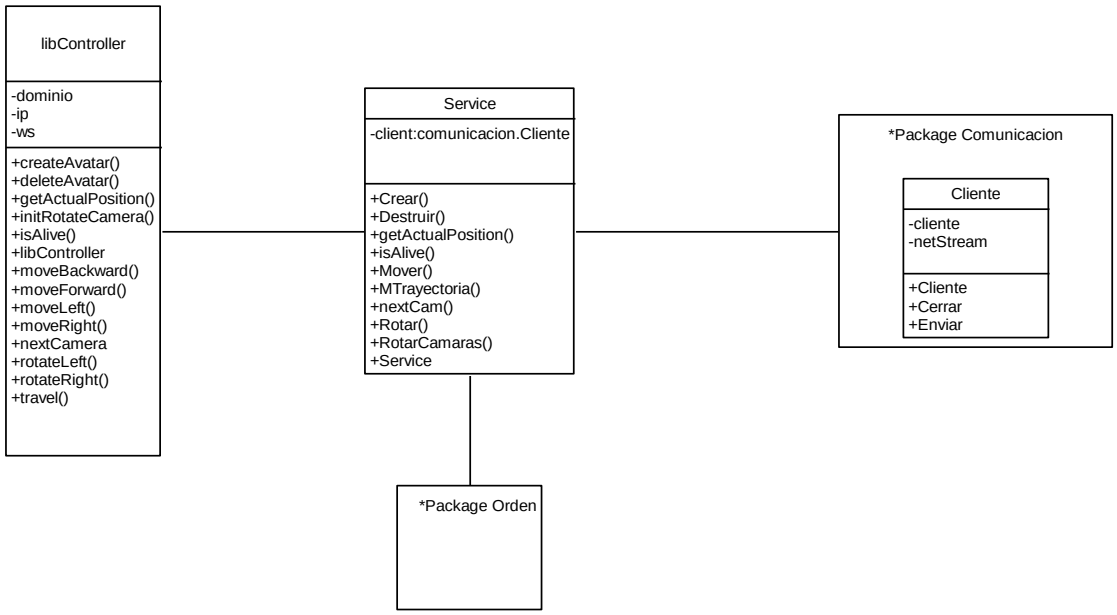
Estos paquetes se han detallado a parte por la importancia que tienen para el sistema.

4.6.1. Modelo de clases del paquete Orden



El diagrama ilustra la jerarquía implementada para el tratamiento de órdenes en el servidor. Este conjunto de clases permite encapsular cada tipo de orden en una clase propia para ella, facilitando su envío al servidor. Así mismo facilita la extensibilidad del sistema, ya que permite añadir una nueva orden solo añadiendo una nueva clase y su tratamiento en el servidor.

4.6.2. Modelo de clases del paquete libController



El presente modelo de clases de diseño muestra una librería controladora para la unificación de la lógica de negocio que pueden contener los clientes. Esta librería permite abstraer la instanciación del web Service, simplificando el uso del mismo.

4.7.Plan de pruebas

4.7.1. Pruebas unitarias

Las pruebas unitarias se realizan sobre cada una de las partes estructurales de la aplicación de forma que se realizan pruebas de límites para cada campo de las clases que intervienen en el diseño, tratamiento de errores de cada método, así como asegurarse de que los campos que emplea cada uno de los formularios acepta los tipos de datos para los que están pensados.

Es importante señalar que las pruebas unitarias no descubrirán todos los errores del código. Los errores de integración del sistema no serán descubiertos, ya que solo se harán las pruebas sobre los distintos módulos de código. Tampoco se descubrirán errores de rendimiento, ni errores que afecten a todo el sistema.

4.7.2. Pruebas de integración

Las pruebas de integración son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez.

Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos.

Los puntos de aplicación de las pruebas de integración han sido cada uno de los módulos del sistema. Se han realizado pruebas de integración sobre el proyecto que carga el servidor, probando la recepción y tratamiento adecuado de cada una de las órdenes. Así

mismo se han realizado las pruebas correspondientes al sistema que se encarga de realizar el render de los modelos y escenarios. Se han hecho pruebas por separado sobre los movimientos de los personajes, la carga de distintos modelos para cada personaje así como los movimientos de las cámaras. Tras esto se procedió a integrar con el servidor, probando que las órdenes recibidas.

Una vez probada toda la parte del servidor se ha probado el funcionamiento del web service, invocando cada uno de sus métodos y probando su efecto en el servidor, se ha comprobado que las acciones que se emiten son las correspondientes y que el servidor responde adecuadamente a las mismas.

Finalmente se realizaron las pruebas a los clientes, primero probando la librería que hace las funciones de controlador (libController). Posteriormente se integró la librería con las interfaces gráficas, probando que cada uno de los controles que contienen cumplen con su función y que el servidor responde correctamente y actúa en consecuencia a las órdenes enviadas por los clientes.

4.7.3. Pruebas del sistema

Las pruebas de sistema son aquellas que han sido realizadas sobre el conjunto global de la aplicación. Las pruebas de sistema tratan de simular lo que sería el uso normal de la aplicación en un entorno de producción para descubrir así los posibles errores.

Las pruebas sobre el sistema han consistido en lanzar múltiples clientes simultáneos en distintas máquinas y en la misma máquina probando hasta qué punto es escalable el sistema.

Para ello se ha realizado una modificación sobre los clientes que procesan los ficheros. En lugar de utilizar una interfaz gráfica, se empleará un cliente en modo consola, que cumpla las mismas funciones que el cliente con GUI.

Así mismo se han realizado pruebas sobre distintas arquitecturas hardware, tales como las que se describen a continuación:

- La primera arquitectura consiste en lanzar tanto clientes como servidor como servicio web en la misma máquina.
- La segunda arquitectura consiste en lanzar los clientes en distintas máquinas clientes, mientras que servidor y servicio web residen en la misma máquina. La comunicación entre clientes y servicio web puede ser o bien en red local o bien a través de internet.
- La última de las arquitecturas probadas está formada por múltiples máquinas clientes, un servidor web en el que reside el web Service y finalmente una máquina independiente en la que se ejecuta la aplicación de servidor. La comunicación entre clientes y servicio web puede ser o bien en red local o bien a través de internet, mientras que la comunicación entre web Service y servidor se realiza en red local exclusivamente, aunque podría realizarse a través de internet.

Las pruebas consistieron en lanzar los clientes, aumentando el número de clientes simultáneos de uno en uno. Para empezar se ejecuta un solo cliente. Después dos, a continuación tres y así sucesivamente.

Para la primera arquitectura se observó que a partir de 3 clientes simultáneos empezaba a dar problemas de rendimiento, ya que no era capaz de procesar las peticiones del web Service a la velocidad adecuada, perdiéndose algunas de éstas, por lo que los personajes no completaban los movimientos de manera adecuada o ni siquiera se llegaban a crear. Este problema solo se produce cuando todos los clientes se ejecutan prácticamente a la vez. En el

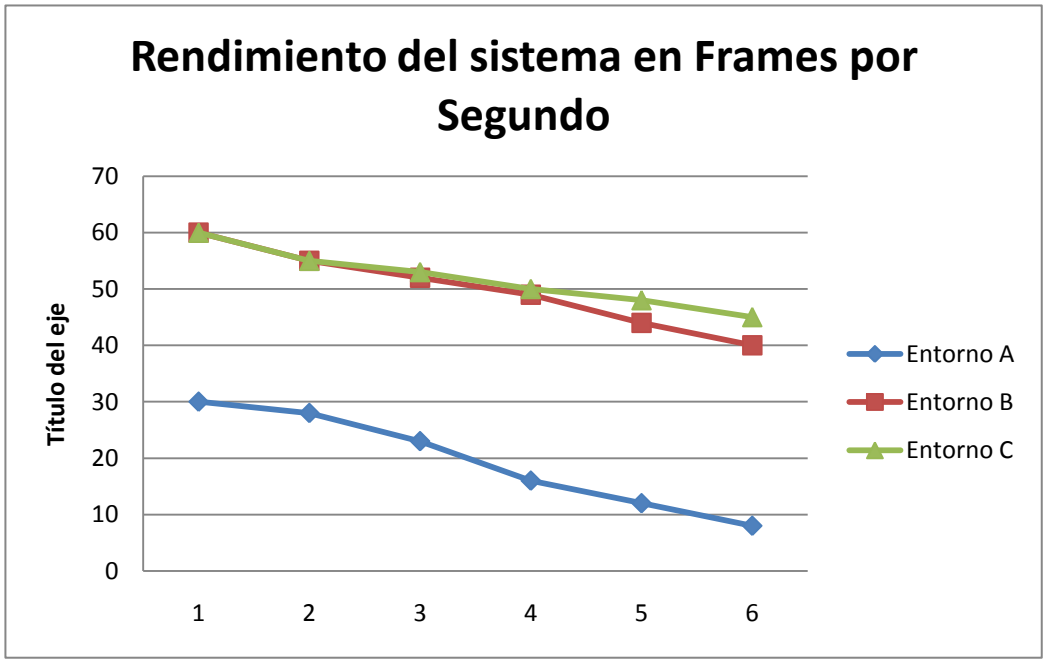
caso de que se lancen los clientes de manera escalonada no aparecen problemas de rendimiento. Así mismo, no se observan pérdidas de rendimiento del motor gráfico a la hora de procesar los múltiples modelos que se van creando a medida que se emplean más clientes.

Para la segunda arquitectura se lanzaron los clientes en tres máquinas clientes distintas, de manera escalonada, al igual que se hizo anteriormente. A medida que se incrementaba el número de clientes, se observa como el servidor va perdiendo rendimiento debido a la carga de procesado de peticiones por un lado y del sistema de renderizado por el otro. Hasta seis clientes se comporta de manera adecuada, siendo la perdida de rendimiento muy baja. Ya que el prototipo no dispone de capacidad de generar más de 6 clientes no se han realizado pruebas con más clientes, pero según las pruebas realizadas podría escalar bien hasta unos diez clientes, basándonos en los Frames por segundo que genera la aplicación servidora.

Finalmente en la tercera arquitectura se han empleado tres máquinas para lanzar clientes, un servidor web con Windows server 2008 y un servidor con Windows Vista Profesional Edition para la generación de los entornos, resultando en una arquitectura completamente distribuida. El número de clientes se ha ido aumentando paulatinamente, al igual que se hizo en los casos anteriores, sin observar pérdidas de rendimiento al lanzar hasta seis clientes simultáneos.

La siguiente tabla ilustra el rendimiento en frames por segundo (FPS) del sistema según las pruebas realizadas con cada uno de los entornos. Se entiende como Entorno A, el entorno en el que todo el sistema se ejecuta en la misma máquina. El entorno B es aquel en el que la ejecución se hace servidor y servicio web en una máquina y clientes en otra y Entorno C en el que tanto clientes como servidor como servicio Web se encuentran cada uno en una máquina diferente.

Clientes	Entorno A	Entorno B	Entorno C
1	30 FPS	60 FPS	60 FPS
2	28 FPS	55 FPS	55 FPS
3	23 FPS	52 FPS	53 FPS
4	16 FPS	49 FPS	50 FPS
5	12 FPS	44 FPS	48 FPS
6	8 FPS	40 FPS	45 FPS



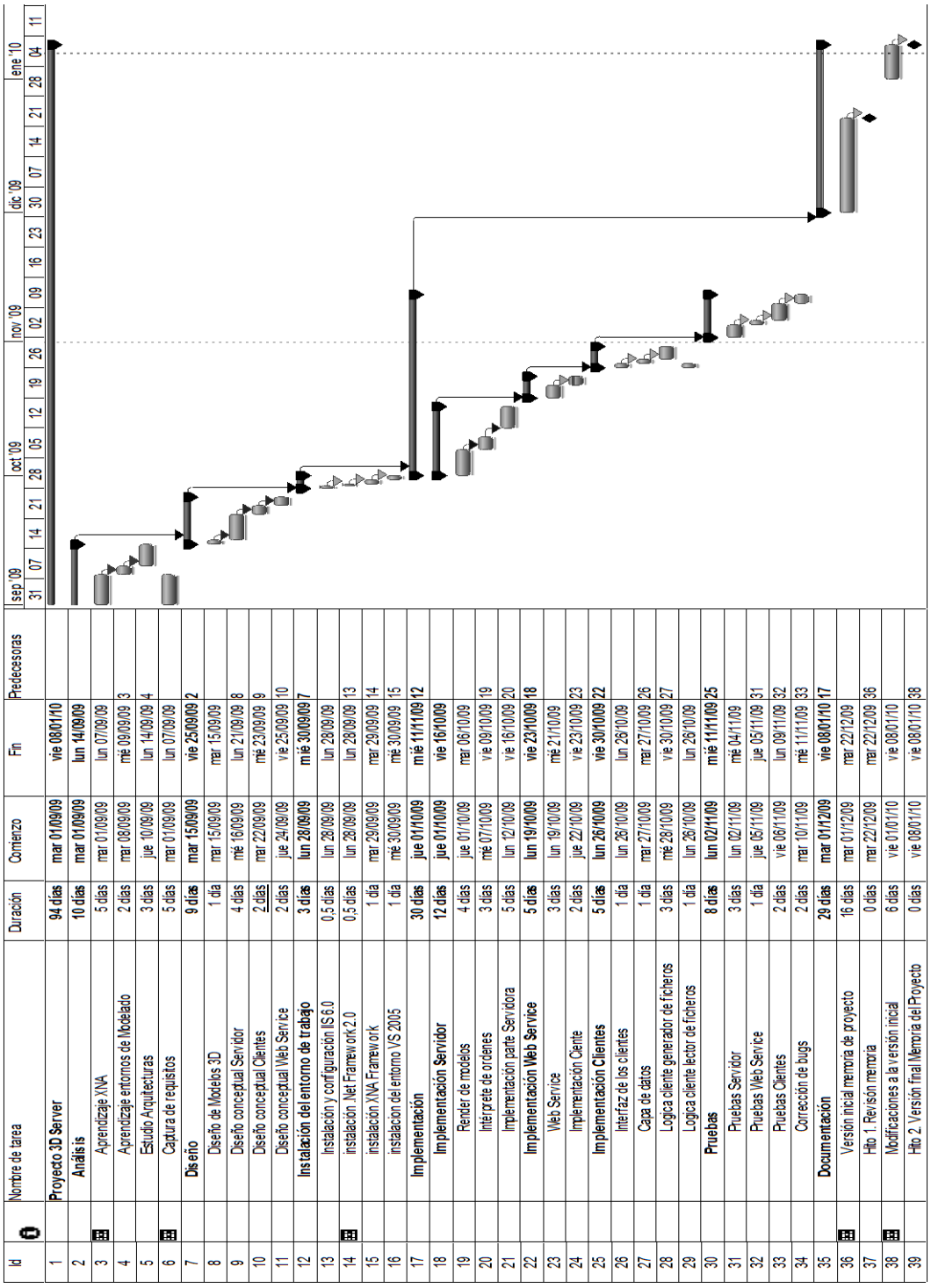
Como se observa en el Gráfico el entorno con un rendimiento menor es el A, ya que los clientes constan de bucles que consumen una gran cantidad de recursos, que junto con las tareas de renderizado hacen que el rendimiento decrezca. En cuanto a los entornos B y C, tienen un rendimiento similar al tener distribuidos los clientes en distintas máquinas, así como disponer de un servidor para realizar el renderizado de manera exclusiva.

El entorno C es el más recomendado para el sistema actual ya que ofrece un mayor rendimiento al realizar la separación entre servidor de render y servidor web de manera que se disponen de máquinas dedicadas a cada uno de los distintos sistemas.

5. Estudio de Viabilidad.

A continuación se muestra un estudio de viabilidad del desarrollo de este proyecto en el cual se muestran las distintas fases del desarrollo, así como la duración de las mismas, acompañado de un diagrama Gantt. Finalmente se muestra un presupuesto para el presente proyecto.

5.1. Plan de proyecto.



Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Proyecto 3D Server**	**94 días**	**mar 01/09/09**	**vie 08/01/10**	
2	**Análisis**	**10 días**	**mar 01/09/09**	**lun 14/09/09**	
3	Aprendizaje XNA	5 días	mar 01/09/09	lun 07/09/09	
4	Aprendizaje entornos de Modelado	2 días	mar 08/09/09	mié 09/09/09	3
5	Estudio Arquitecturas	3 días	jue 10/09/09	lun 14/09/09	4
6	Captura de requisitos	5 días	mar 01/09/09	lun 07/09/09	
7	**Diseño**	**9 días**	**mar 15/09/09**	**vie 25/09/09**	**2**
8	Diseño de Modelos 3D	1 día	mar 15/09/09	mar 15/09/09	
9	Diseño conceptual Servidor	4 días	mié 16/09/09	lun 21/09/09	8
10	Diseño conceptual Clientes	2 días	mar 22/09/09	mié 23/09/09	9
11	Diseño conceptual Web Service	2 días	jue 24/09/09	vie 25/09/09	10
12	**Instalación del entorno de trabajo**	**3 días**	**lun 28/09/09**	**mié 30/09/09**	**7**
13	Instalación y configuración IIS 6.0	0,5 días	lun 28/09/09	lun 28/09/09	
14	instalación .Net Framework 2.0	0,5 días	lun 28/09/09	lun 28/09/09	13
15	instalación XNA Framework	1 día	mar 29/09/09	mar 29/09/09	14
16	instalacion del entorno VS 2005	1 día	mié 30/09/09	mié 30/09/09	15
17	**Implementación**	**30 días**	**jue 01/10/09**	**mié 11/11/09**	**12**
18	**Implementación Servidor**	**12 días**	**jue 01/10/09**	**vie 16/10/09**	
19	Render de modelos	4 días	jue 01/10/09	mar 06/10/09	
20	Intérprete de ordenes	3 días	mié 07/10/09	vie 09/10/09	19
21	Implementación parte Servidora	5 días	lun 12/10/09	vie 16/10/09	20
22	**Implementación Web Service**	**5 días**	**lun 19/10/09**	**vie 23/10/09**	**18**
23	Web Service	3 días	lun 19/10/09	mié 21/10/09	
24	Implementación Cliente	2 días	jue 22/10/09	vie 23/10/09	23
25	**Implementación Clientes**	**5 días**	**lun 26/10/09**	**vie 30/10/09**	**22**
26	Interfaz de los clientes	1 día	lun 26/10/09	lun 26/10/09	
27	Capa de datos	1 día	mar 27/10/09	mar 27/10/09	26
28	Logica cliente generador de ficheros	3 días	mié 28/10/09	vie 30/10/09	27
29	Logica cliente lector de ficheros	1 día	lun 26/10/09	lun 26/10/09	
30	**Pruebas**	**8 días**	**lun 02/11/09**	**mié 11/11/09**	**25**
31	Pruebas Servidor	3 días	lun 02/11/09	mié 04/11/09	
32	Pruebas Web Service	1 día	jue 05/11/09	jue 05/11/09	31
33	Pruebas Clientes	2 días	vie 06/11/09	lun 09/11/09	32
34	Corrección de bugs	2 días	mar 10/11/09	mié 11/11/09	33
35	**Documentación**	**29 días**	**mar 01/12/09**	**vie 08/01/10**	**17**
36	Versión inicial memoria de proyecto	16 días	mar 01/12/09	mar 22/12/09	
37	Hito 1: Revisión memoria	0 días	mar 22/12/09	mar 22/12/09	36
38	Modificaciones a la versión inicial	6 días	vie 01/01/10	vie 08/01/10	
39	Hito 2: Versión final Memoria del Proyecto	0 días	vie 08/01/10	vie 08/01/10	38

Como complemento al diagrama GANT de la planificación del proyecto, se adjunta una tabla con todas las tareas detalladas.

Proyecto 3D Server	**94 días**	**01/09/2009**	**08/01/2010**	
Análisis	**10 días**	**01/09/2009**	**14/09/2009**	
Aprendizaje XNA	5 días	01/09/2009	07/09/2009	
Aprendizaje entornos de Modelado	2 días	08/09/2009	09/09/2009	3
Estudio Arquitecturas	3 días	10/09/2009	14/09/2009	4
Captura de requisitos	5 días	01/09/2009	07/09/2009	
Diseño	**9 días**	**15/09/2009**	**25/09/2009**	**2**
Diseño de Modelos 3D	1 día	15/09/2009	15/09/2009	
Diseño conceptual Servidor	4 días	16/09/2009	21/09/2009	8
Diseño conceptual Clientes	2 días	22/09/2009	23/09/2009	9
Diseño conceptual Web Service	2 días	24/09/2009	25/09/2009	10
Instalación del entorno de trabajo	**3 días**	**28/09/2009**	**30/09/2009**	**7**
Instalación y configuración IIS 6.0	0,5 días	28/09/2009	28/09/2009	
instalación .Net Framework 2.0	0,5 días	28/09/2009	28/09/2009	13
instalación XNA Framework	1 día	29/09/2009	29/09/2009	14
instalacion del entorno VS 2005	1 día	30/09/2009	30/09/2009	15
Implementación	**30 días**	**01/10/2009**	**11/11/2009**	**12**
Implementación Servidor	**12 días**	**01/10/2009**	**16/10/2009**	
Render de modelos	4 días	01/10/2009	06/10/2009	

Intérprete de ordenes	3 días	07/10/2009	09/10/2009	19
Implementación parte Servidora	5 días	12/10/2009	16/10/2009	20
Implementación Web Service	**5 días**	**19/10/2009**	**23/10/2009**	**18**
Web Service	3 días	19/10/2009	21/10/2009	
Implementación Ciente	2 días	22/10/2009	23/10/2009	23
Implementación Clientes	**5 días**	**26/10/2009**	**30/10/2009**	**22**
Interfaz de los clientes	1 día	26/10/2009	26/10/2009	
Capa de datos	1 día	27/10/2009	27/10/2009	26
Logica cliente generador de ficheros	3 días	28/10/2009	30/10/2009	27
Logica cliente lector de ficheros	1 día	26/10/2009	26/10/2009	
Pruebas	**8 días**	**02/11/2009**	**11/11/2009**	**25**
Pruebas Servidor	3 días	02/11/2009	04/11/2009	
Pruebas Web Service	1 día	05/11/2009	05/11/2009	31
Pruebas Clientes	2 días	06/11/2009	09/11/2009	32
Corrección de bugs	2 días	10/11/2009	11/11/2009	33
Documentación	**29 días**	**01/12/2009**	**08/01/2010**	**17**
Versión inicial memoria de proyecto	16 días	01/12/2009	22/12/2009	
Hito 1. Revisón memoria	**0 días**	**22/12/2009**	**22/12/2009**	**36**
Modificaciones a la versión inicial	6 días	01/01/2010	08/01/2010	
Hito 2. Versión final Memoria del Proyecto	**0 días**	**08/01/2010**	**08/01/2010**	**38**

5.2.Estimación del desarrollo.

Para realizar la estimación del proyecto, se empleará el método COCOMO II, que permite obtener una estimación del tiempo de desarrollo y del tamaño del mismo.

Para ello, habrá que determinar qué tipo de operación hace cada funcionalidad de la aplicación (entrada, salida o consulta) y determinar su complejidad, en función de los registros y parámetros de cada funcionalidad, así como los ficheros externos que utiliza.

TAREA	TIPO	det	Ftr	COMPLEJIDAD
SERVIDOR				
ArrancaJuego	**Entrada**	**0**	**1**	**Baja**
ProcesarOrden	**Entrada**	**2**	**2**	**Baja**
ProcesarLinea	**Entrada**	**2**	**2**	**Baja**
Camara	**Entrada**	**2**	**1**	**Baja**
ActivarRotacionCamara	**Entrada**	**1**	**1**	**Baja**
addAvatar	**Entrada**	**1**	**1**	**Baja**
addOrden	**Entrada**	**1**	**9**	**Baja**
colisionAvatars	**Consulta**	**1**	**1**	**Media**
colisionTerreno	**Consulta**	**1**	**1**	**Media**
createAvatar	**Entrada**	**3**	**3**	**Baja**
destroyAvatar	**Entrada**	**1**	**1**	**Baja**
Draw	**Entrada**	**1**	**1**	**Alta**
DrawGameObject	**Entrada**	**1**	**1**	**Alta**
DrawTerrain	**Entrada**	**1**	**1**	**Alta**

getActualPosition	**Salida**	**1**	**1**	**Baja**
getOrden	**Consulta**	**0**	**1**	**Baja**
isAlive	**Consulta**	**1**	**1**	**Baja**
ModificarCamara	**Entrada**	**0**	**1**	**Baja**
MoveAvatar	**Entrada**	**2**	**2**	**Alta**
nextCamera	**Entrada**	**0**	**1**	**Baja**
rotateAvatar	**Entrada**	**2**	**2**	**Media**
separarMovimientoTrayectoria	**Entrada**	**1**	**1**	**Alta**
Update	**Entrada**	**1**	**1**	**Media**
updateAvatarPosition	**Entrada**	**0**	**1**	**Alta**
UpdateCamera	**Entrada**	**0**	**1**	**Media**

CLIENTES				
procesarLinea	**Entrada**	**1**	**1**	**Baja**
procesaOrden	**Entrada**	**1**	**1**	**Baja**
ArrancaJuego	**Entrada**	**0**	**1**	**Baja**
Movimiento	**Entrada**	**2**	**2**	**Baja**
Rotar	**Entrada**	**2**	**2**	**Baja**
GuardarEnFichero	**Salida**	**1**	**1**	**Media**
SiguienteCamara	**Entrada**	**0**	**1**	**Baja**
Trayectoria	**Entrada**	**4**	**3**	**Media**
IntervaloCamara	**Entrada**	**1**	**1**	**Baja**

SERVICIO WEB				
Crear	**Entrada**	**7**	**2**	**Baja**

Destruir	**Entrada**	**3**	**1**	**Baja**
getActualPosition	**Entrada**	**3**	**1**	**Baja**
isAlive	**Entrada**	**3**	**1**	**Baja**
Mover	**Entrada**	**6**	**2**	**Baja**
MtTrayectoria	**Entrada**	**7**	**2**	**Baja**
nextCam	**Entrada**	**0**	**1**	**Baja**
Rotar	**Entrada**	**4**	**2**	**Baja**
rotarCamaras	**Entrada**	**1**	**1**	**Baja**

Adicionalmente, para cada entrada existe una salida que devuelve una confirmación indicando el estado en el que ha finalizado la funcionalidad.

Resumen:

	Servidor	**Clientes**	**Servicio web**
Entradas	**20**	**8**	**9**
Salidas	**21**	**9**	**9**
Consultas	**4**	**0**	**0**

Project Name: Render System

Scale Factor | Schedule

Development Model: Post Architecture

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	Servidor Rende	F:2140	0.00	0.12	Object-Orient	7.2	0.8	2535.6	0.00	0.0	0.2	0.0
	Cliente	F:770	0.00	0.11	Object-Orient	2.6	0.3	2850.1	0.00	0.0	0.1	0.0
	Web_service	F:700	0.00	0.10	Object-Orient	2.3	0.2	2855.9	0.00	0.0	0.1	0.0

Total Lines of Code: 3610

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	1.1	3.8	3319.8	0.00	0.0	0.3	
Most Likely	1.4	4.0	2655.9	0.00	0.0	0.3	0.0
Pessimistic	1.7	4.3	2124.7	0.00	0.0	0.4	

Como se puede ver en la estimación realizada mediante el método COCOMO II, y ajustado el factor de forma que se señale un alto conocimiento en el uso del lenguaje de desarrollo, ya que se ha estado empleando durante un tiempo), la complejidad de los datos, así como parámetros de concurrencia y demás, se prevé que el desarrollo se lleve a cabo en un mes y medio con un desarrollador.

Una vez realizado el desarrollo se ha visto que el desarrollo ha durado 30 días laborables, para el cual en vez de un desarrollador, ha participado un ingeniero júnior.

5.3.Presupuesto.

Recursos Humanos: el único recurso humano que ha intervenido en el desarrollo del proyecto es un ingeniero *júnior*.

- Análisis: 80 horas.

- Diseño: 72 horas.

- Instalación del entorno de trabajo (servidores incluidos): 24 horas.

- Implementación y pruebas: 240 horas.

- Documentación: 232 horas.

Con un precio total por hora de un ingeniero júnior de 60 €/hora.

Tarea	Horas	Precio total
Análisis	**80**	**4.800,00 €**
Diseño	**72**	**4.320,00 €**
Instalación y configuración del entorno de trabajo (servidores)	**24**	**1.440,00 €**
Implementación y pruebas	**240**	**14.400,00 €**
Documentación	**232**	**13.920,00 €**
Total	**648**	**38.880,00 €**

Concepto	Precio	Iva (16%)	total
Material Hardware			
Servidor	**800,00 €**	**128,00 €**	**928,00 €**
Portatil desarrollo	**700,00 €**	**112,00 €**	**812,00 €**
material Software			
Licencia Visual Studio	**690,00 €**	**110,40 €**	**800,40 €**
Licencia Windows server 2003	**2.200,00 €**	**352,00 €**	**2.552,00 €**
Licencia SQL Server 2005	**2.300,00 €**	**368,00 €**	**2.668,00 €**
Recursos Humanos	**38.880,00 €**	**6.220,80 €**	**45.100,80 €**
Total			**52.861,20 €**

El presupuesto asciende a **Cincuenta y dos mil ochocientos sesenta y uno con veinte Euros**.

Colmenarejo, Enero de 2010.

El Ingeniero Autor del Proyecto.

Fdo. José Miguel Colmena de Celis.

6. Despliegue y Uso del prototipo

En este punto se describirá como realizar una instalación de los distintos clientes, del servidor y los distintos clientes, así como realizar el despliegue del web service en el servidor web desde Visual Studio.

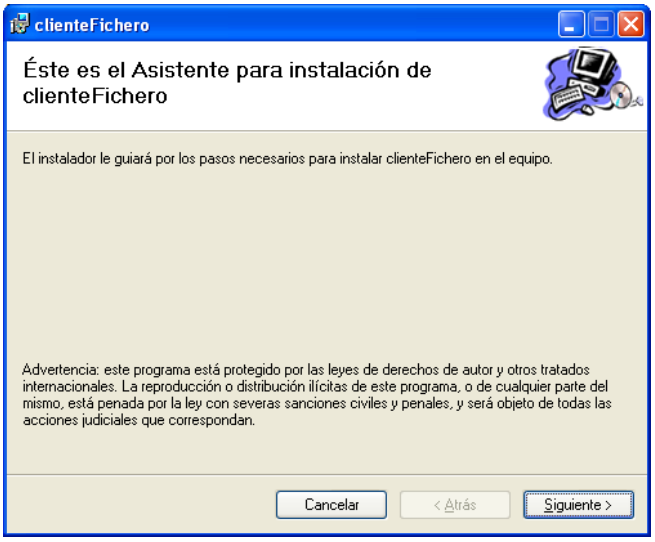
Posteriormente se mostrará un pequeño manual de usuario que ilustra a grandes rasgos como emplear el prototipo desarrollado.

6.1.Despliegue del prototipo

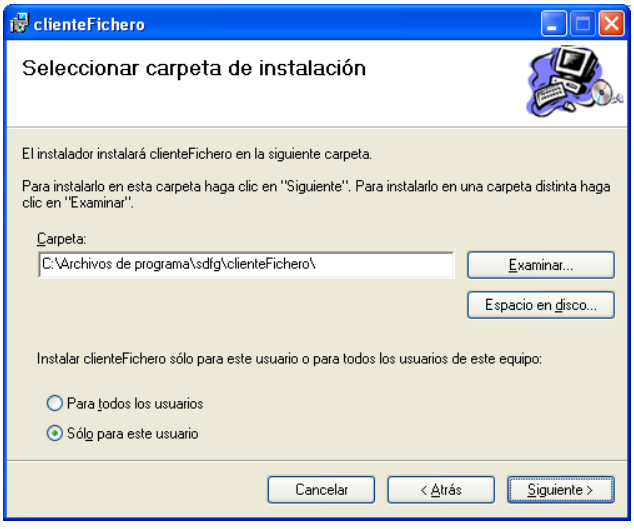
Para el despliegue del prototipo se han generado 3 instaladores, dos corresponden a los clientes y el tercero al servidor. Mediante estos instaladores queda todo configurado para empezar a usar el prototipo.

6.1.1. Instalación del cliente lector de ficheros

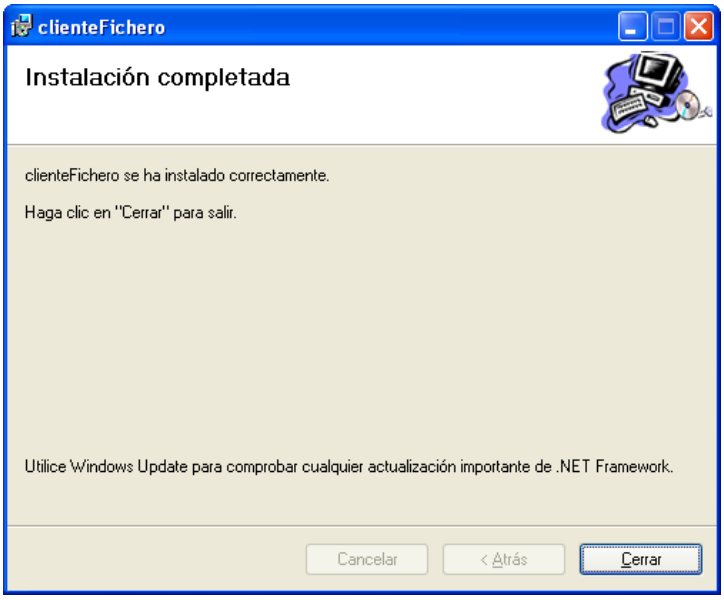
Para realizar la instalación del cliente lector de ficheros bastará con ejecutar el archivo llamado “clienteFichero.msi” y seguiremos el asistente.



Una vez iniciado el instalador, pulsaremos siguiente en esta primera ventana de dialogo.



En esta otra ventana seleccionaremos el directorio de instalación e indicaremos que usuarios podrán usar el cliente. Una vez completado pulsaremos el botón Siguiente. En la siguiente ventana que aparezca se volverá a pulsar el botón Siguiente.

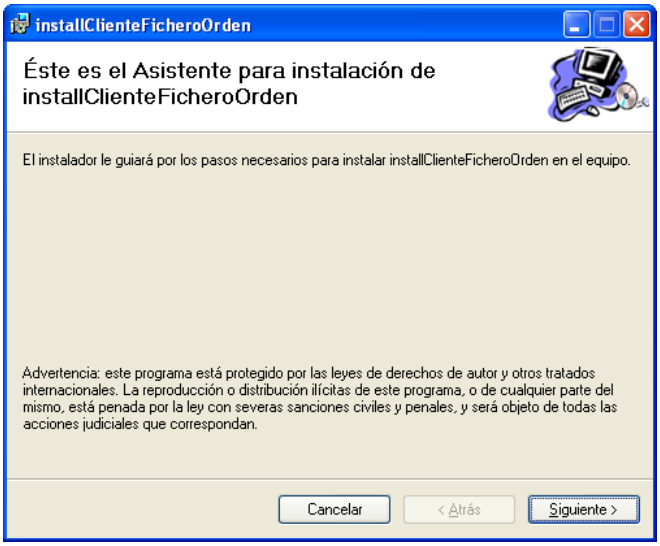


Una vez finaliza la instalación pulsaremos el botón Cerrar y para comprobar que todo ha ido bien iremos al escritorio de Windows a buscar el acceso directo llamado "ClienteFichero". Haciendo doble click sobre éste, se ejecutará el cliente y se podrá empezar a trabajar con él.

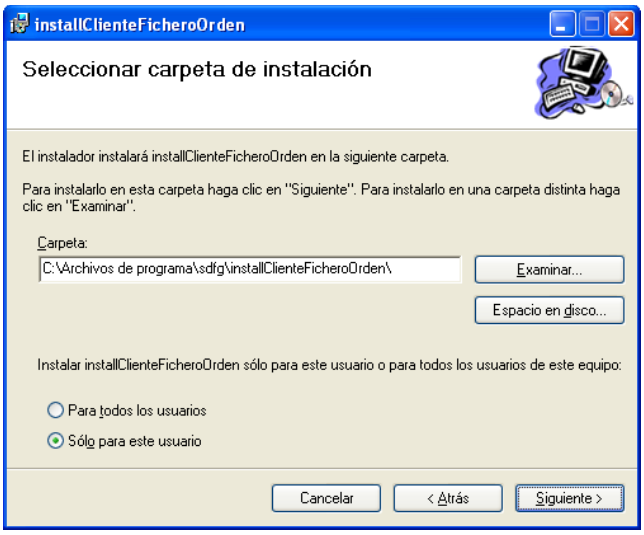
Puesto que los proyectos han sido desarrollados con .Net Framework, si el ordenador donde se pretenden instalar no tuviera la versión adecuada, .Net Framework 2.0, así como la versión adecuada del programa Microsoft Windows Installer, el propio instalador los instalará antes que el programa, ya que de otro modo, no será posible utilizar la aplicación.

6.1.2. Instalación del cliente generador de ficheros

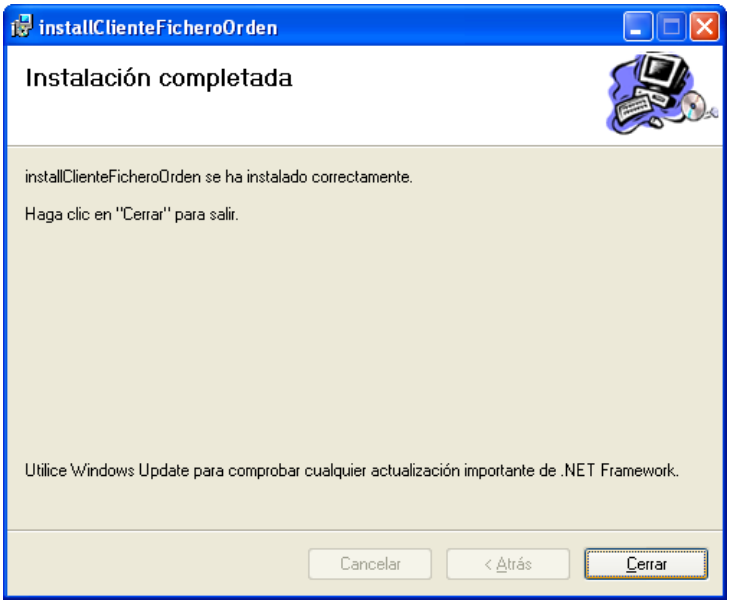
Para realizar la instalación del cliente generador de ficheros bastará con ejecutar el archivo llamado "installClienteFicheroOrden.msi" y seguiremos el asistente.



Una vez iniciado el instalador, pulsaremos siguiente en esta primera ventana de dialogo.



En esta otra ventana seleccionaremos el directorio de instalación e indicaremos que usuarios podrán usar el cliente. Una vez completado pulsaremos el botón Siguiente. En la siguiente ventana que aparezca se volverá a pulsar el botón Siguiente.

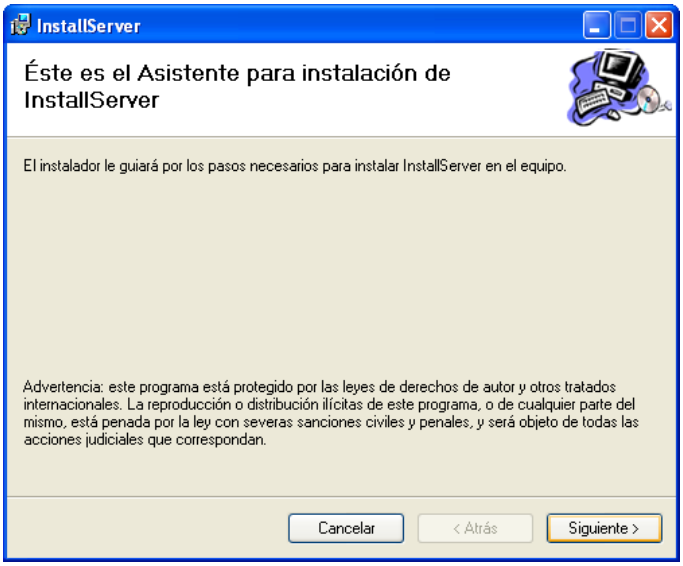


Una vez finaliza la instalación pulsaremos el botón Cerrar y para comprobar que todo ha ido bien iremos al escritorio de Windows a buscar el acceso directo llamado "generaFicheroOrden". Haciendo doble click sobre éste, se ejecutará el cliente y se podrá empezar a trabajar con él.

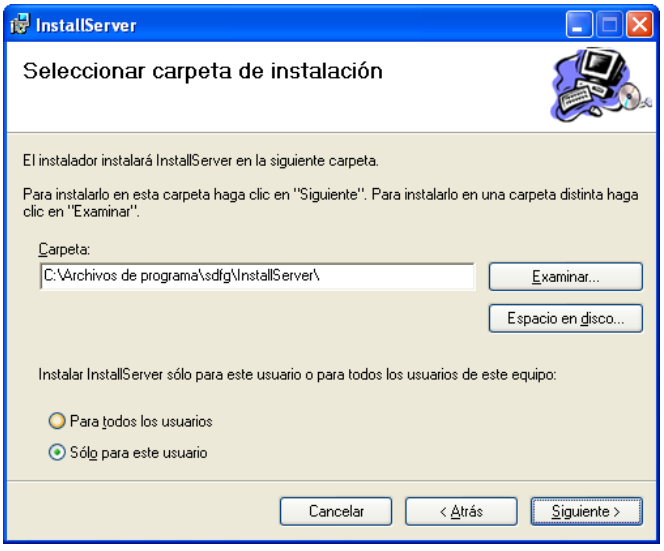
Puesto que los proyectos han sido desarrollados con .Net Framework, si el ordenador donde se pretenden instalar no tuviera la versión adecuada, .Net Framework 2.0, así como la versión adecuada del programa Microsoft Windows Installer, el propio instalador los instalará antes que el programa, ya que de otro modo, no será posible utilizar la aplicación.

6.1.3. Instalación del cliente generador de ficheros

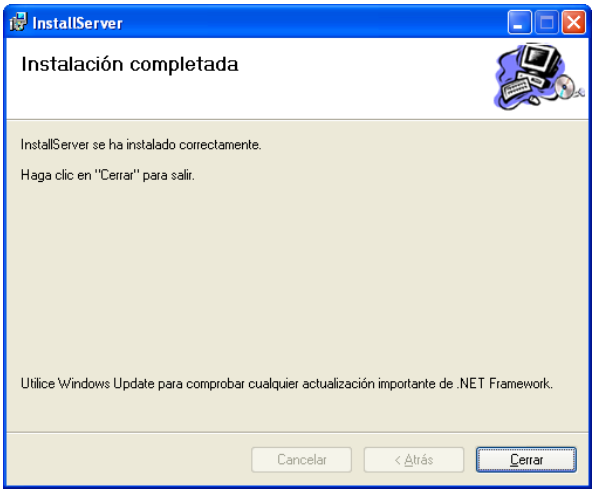
Para realizar la instalación del servidor bastará con ejecutar el archivo llamado "InstallServer.msi" y seguiremos el asistente.



Una vez iniciado el instalador, pulsaremos siguiente en esta primera ventana de dialogo.



En esta otra ventana seleccionaremos el directorio de instalación e indicaremos que usuarios podrán usar el cliente. Una vez completado pulsaremos el botón Siguiente. En la siguiente ventana que aparezca se volverá a pulsar el botón Siguiente.

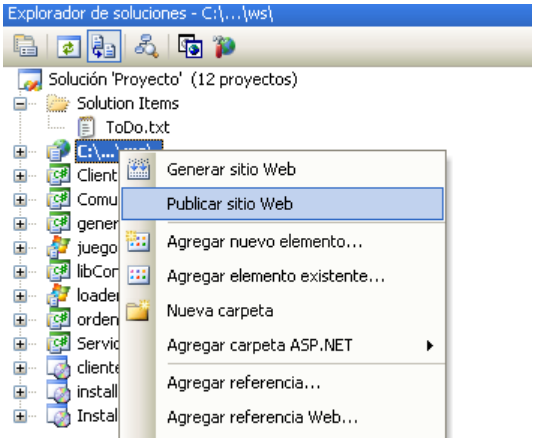


Una vez finaliza la instalación pulsaremos el botón Cerrar y para comprobar que todo ha ido bien iremos al escritorio de Windows a buscar el acceso directo llamado "ServidorJuego3D". Haciendo doble click sobre éste, se ejecutará el cliente y se podrá empezar a trabajar con él.

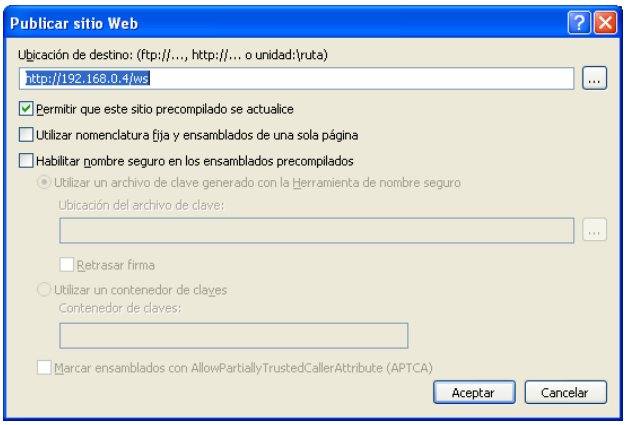
Puesto que los proyectos han sido desarrollados con .Net Framework, si el ordenador donde se pretenden instalar no tuviera la versión adecuada, .Net Framework 2.0, así como la versión adecuada del programa Microsoft Windows Installer, el propio instalador los instalará antes que el programa, ya que de otro modo, no será posible utilizar la aplicación.

6.1.4. Despliegue del web service

Para realizar el despliegue del web service habrá que pulsar con el botón derecho del en el ítem correspondiente al web service en el explorador de soluciones de Visual Studio y seleccionar la opción Publicar sitio Web.



Una vez seleccionada la opción Publicar sitio Web aparecerá una ventana en la que seleccionar el destino. Seleccionaremos un servidor, bien sea en local o bien sea en un servidor remoto y pulsaremos el botón “Aceptar”.



Finalmente, el sistema podría pedir un usuario y contraseña del servidor remoto para poder publicar el web service en el lugar indicado. Hecho esto quedará desplegado el Web Service y se podrá acceder al él. Puesto que el sistema está en fase de prototipo, habrá que actualizar la referencia al Web Service desde Visual Studio antes de generar los instaladores.

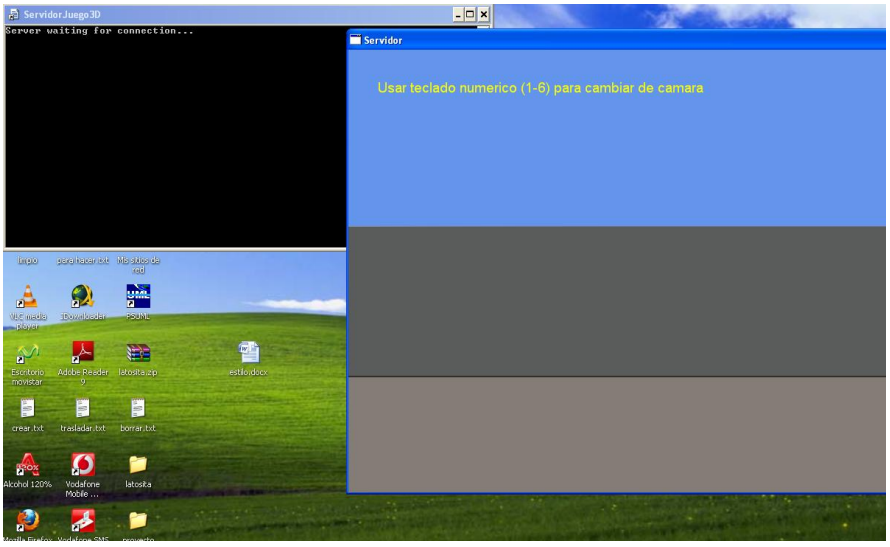
6.2.Manual de usuario

En esta sección se mostrará el uso, a grandes rasgos del prototipo implementado.

6.2.1. Servidor

En primer lugar se detallará el uso del servidor, ya que para poder emplear el prototipo debe estar en ejecución el servidor, así como el web service que habrá sido previamente desplegado desde Visual Studio 2005.

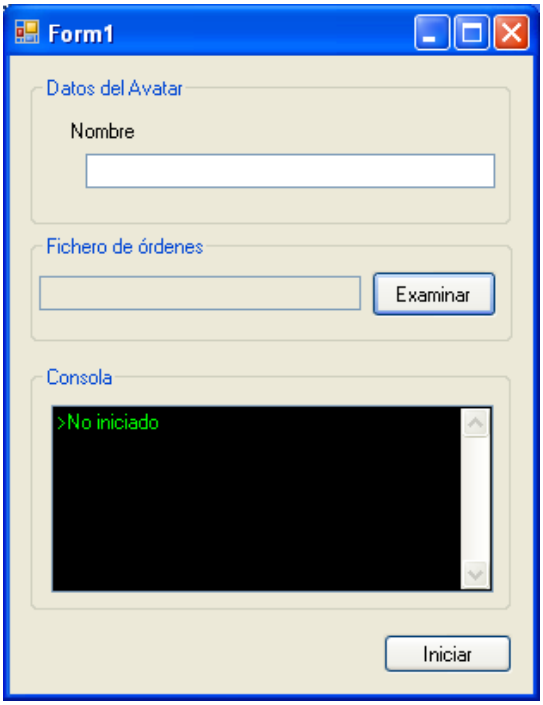
Para lanzar el servidor se hará doble click en el acceso directo que se encuentra en el escritorio con el nombre “ServidorJuego3D” con ello se abrirá la consola del servidor y la ventana de render:



El usuario podrá, mediante el teclado, numérico, la vista de la cámara en la ventana de render, pero no podrá hacer nada más.

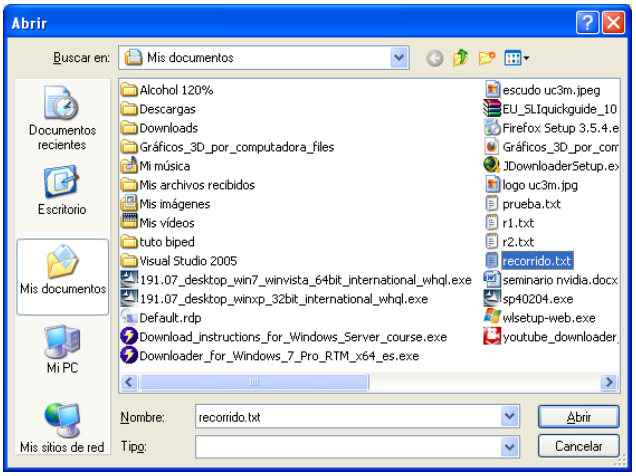
6.2.2. Cliente lector de ficheros

Para lanzar el cliente lector de ficheros se hará doble click en el acceso directo que se encuentra en el escritorio con el nombre “ClienteFichero” con ello se abrirá la ventana del cliente:

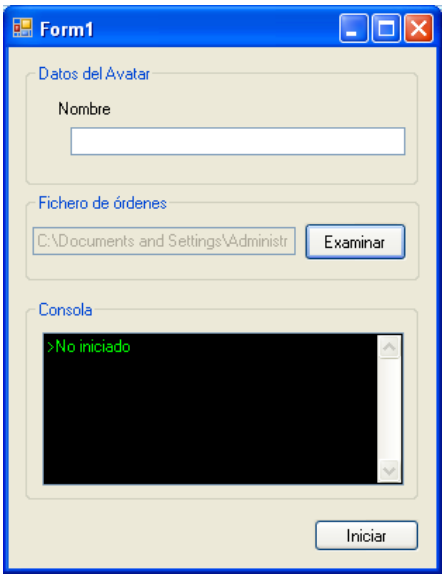


En dicha ventana, el usuario solo podrá introducir un nombre para un avatar en el campo llamado “Nombre”.

Pulsando el botón “Examinar” se abrirá un cuadro de diálogo en el que poder seleccionar un fichero que contenga las órdenes a enviar al servidor.



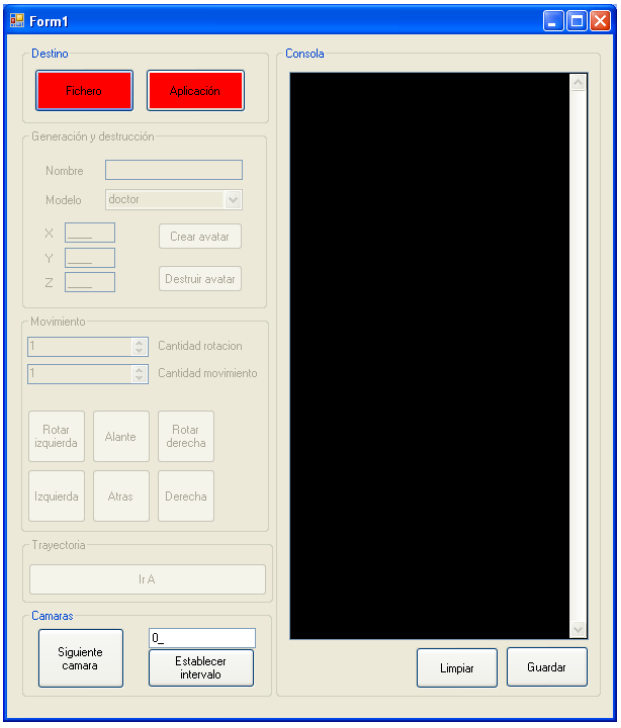
Una vez seleccionado el fichero de texto pulsaremos el botón “Abrir”. En la ventana del cliente, aparecerá la ruta al fichero en el campo de texto que está junto al botón “Examinar”.



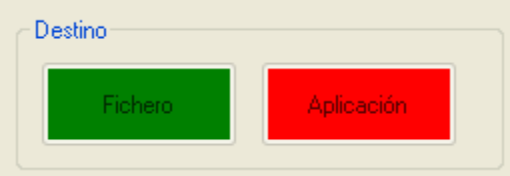
Una vez seleccionado el fichero e indicado un nombre de avatar pulsaremos el botón “iniciar” con lo que el cliente empezará a leer el fichero de texto y enviando las órdenes al Web Service para que las procese y las reenvíe al servidor.

6.2.3. Cliente generador de ficheros

Para lanzar el cliente lector de ficheros se hará doble click en el acceso directo que se encuentra en el escritorio con el nombre “generaFicheroOrden” con ello se abrirá la ventana del cliente:

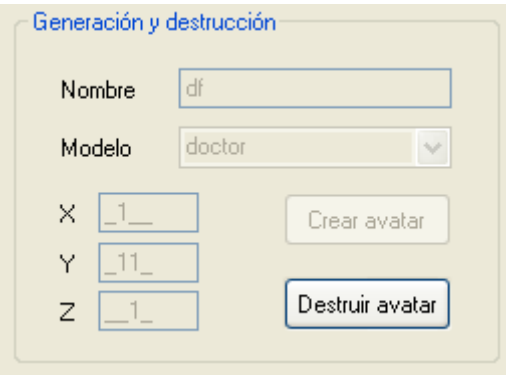


Como se puede observar la ventana está separada en seis partes diferentes. La sección que está a la derecha consta de una lista con las órdenes que se han ido enviando al servidor y se querrán guardar en el fichero. El botón “Limpiar” borra la lista de órdenes. El botón “Guardar” abre un cuadro de diálogo en el que seleccionar la ruta en la cual almacenar el fichero con las órdenes.



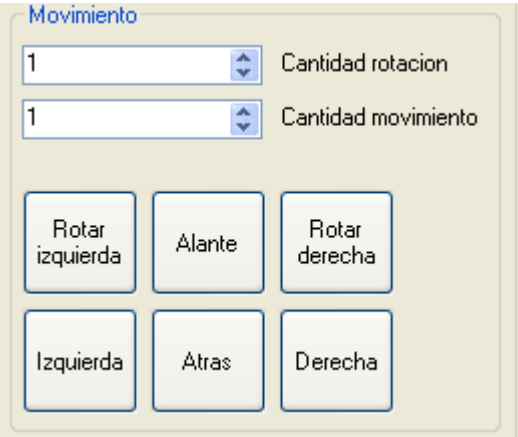
La sección destino permite indicar si solo se querrá dar órdenes sin que se vea una ventana de render con la escena (botón "Fichero"). En este caso, si la aplicación de servidor está ejecutándose, se podrán ver las órdenes que se han lanzado, en caso contrario no se verá nada. Esta funcionalidad nos vale para máquinas en las que no se dispone de una tarjeta gráfica adecuada para lanzar la ventana de render. El segundo caso disponible, permite abrir una ventana con el render de la escena y a medida que se van dando órdenes son mostradas en la ventana de render, permitiendo ver el resultado final de un fichero antes de guardarlo.

A partir de aquí, el resto es común para ambos modos de funcionamiento, a excepción de la sección de cámaras, ya que las órdenes relativas a las cámaras no son almacenadas en el fichero de órdenes, con lo que solo se empleará cuando se haya ejecutado el cliente en modo "Aplicación".

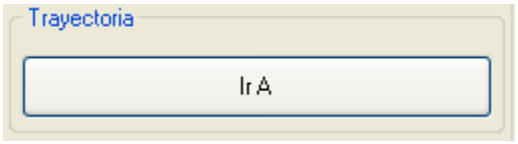


La sección "Generación y destrucción" permite crear y destruir un avatar. Para crear un avatar habrá que, primero introducir un nombre de avatar, a continuación seleccionar un

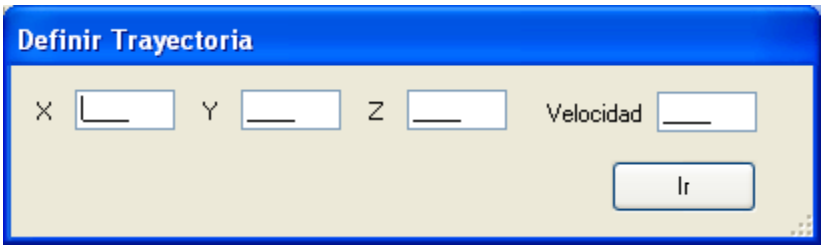
modelo para el avatar, disponiendo de cinco modelos distintos, un doctor y tres soldados, finalmente hay tres campos en los cuales se introducirán las coordenadas de inicio para ese avatar. Una vez se han cumplimentado los campos anteriores, pulsaremos el botón "Crear avatar", con lo que se añadirá dicha orden a la lista y se activarán el botón "Destruir avatar" y las secciones "Movimiento" y "Trayectoria". Mediante el botón "Destruir avatar" se enviará una orden para destruir el avatar cuyo nombre coincide con el nombre especificado en el campo "Nombre" de la sección "Generación y destrucción".



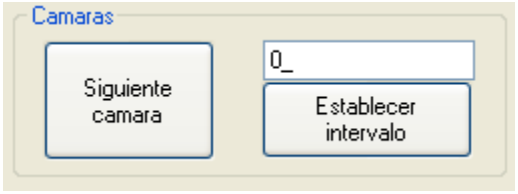
La sección "Movimiento" permite generar distintas órdenes para realizar movimientos. Cada vez que se pulsa uno de los botones se añade la orden correspondiente al botón pulsado. Esto generará una orden de movimiento o rotación, según el botón, de una unidad de magnitud. Si se quiere incrementar la cantidad de movimiento y en lugar de desplazarse o rotar una unidad de magnitud, hacerlo en cantidades mayores, bastará con ajustar los valores deseados en los campos "Cantidad rotación" y "Cantidad movimiento".



La sección "Trayectoria" permite generar una orden de trayectoria, que indicará al avatar correspondiente que deberá ir desde el punto en que se encuentre en ese momento al punto de destino que indique el usuario. Al pulsar el botón "Ir A" se abrirá la siguiente ventana:



En dicha ventana se indicarán las coordenadas X,Y,Z de destino, así como la velocidad a la que se desea realizar el movimiento. Una vez introducidas dichas coordenadas y la velocidad, se pulsará el botón "Ir" se cerrará la ventana "Definir Trayectoria" y se añadirá dicha orden a la lista de órdenes.



Finalmente la sección "Cámaras" solo realizará acciones en el modo "Aplicación" tal como se mencionó anteriormente. Mediante estos controles se puede cambiar la cámara que

está activa en el servidor o bien establecer un valor que permita que la cámara cambie de manera automática cada n segundos.

Pulsando el botón “Siguiente cámara” se envía automáticamente al servidor una orden para que active la cámara siguiente. Esto dependerá de la cámara que este activa en ese momento. Puesto que hay definidas seis cámaras, una vez se ha llegado a la sexta, si se pulsa de nuevo el botón, se pasará a la primera, haciendo un cambio de manera circular.

El botón “Establecer intervalo” permite indicar cada cuanto segundos se desea que se cambie de cámara de manera automática. Bastará con introducir un número de segundos en el campo correspondiente y pulsando el botón “Establecer intervalo” se enviará directamente la orden, sin pasar por el fichero, al servidor y comenzará la rotación de cámaras. Si el número de segundos se establece a 0 se detendrá la rotación de las cámaras.

7. Conclusiones

7.1.Logros del proyecto

Este proyecto ha servido para realizar una introducción en las tecnologías enfocadas al desarrollo de videojuegos multiplataforma como puede ser XNA y como éste gestiona el ciclo de vida de un juego en ejecución.

Desde el punto de vista de la arquitectura de un sistema de información, se han evaluado distintas arquitecturas posibles, analizando las distintas formas de comunicaciones disponibles en proyectos relacionados con XNA e intentando entrelazarlo con tecnologías que están enfocadas a aplicaciones de gestión como son los Web Services [3]. Puesto que para este proyecto se han empleado tanto sockets, como web Services [3], como memoria compartida para tratar con el thread que ejecuta el motor de render, se han adquirido los conceptos necesarios para implantar una arquitectura destinada a un sistema de render distribuido.

Otra de las cosas que me han resultado más interesantes es como XNA facilita la tarea de generación de un videojuego. Hasta ahora era necesario conocer en profundidad el lenguaje C++ y las API's (Application Programmer Interface) de DirectX u OpenGL, y las tareas de tratamiento de modelos 3D eran complejas y tediosas. Actualmente con XNA la tarea queda simplificada hasta el punto de parecerse enormemente a la programación de aplicaciones común. No requiere de amplios conocimientos de programación, pero si son recomendables.

Finalmente este proyecto ha permitido profundizar en el desarrollo de modelos 3D y moverlos por un entorno. Puesto que los movimientos dependen de hacia a dónde mira el personaje se tuvo que realizar un estudio de las transformaciones que hay que realizar sobre las matrices de movimiento para poder realizar los movimientos de manera adecuada.

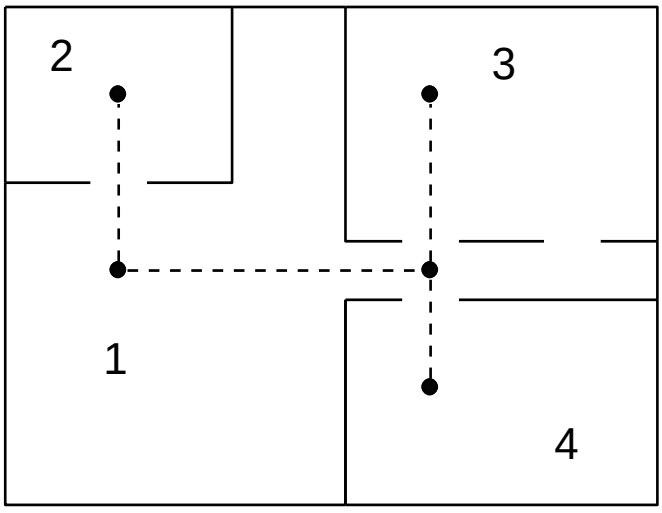
7.2.Futuras mejoras

Este punto del documento trata de las futuras mejoras y capacidades que se podrían implementar en futuras versiones de este proyecto. Algunas de esas mejoras irán encaminadas a ampliar el sistema del servidor y otras estarán enfocadas a los clientes

7.2.1. Mejoras en el servidor

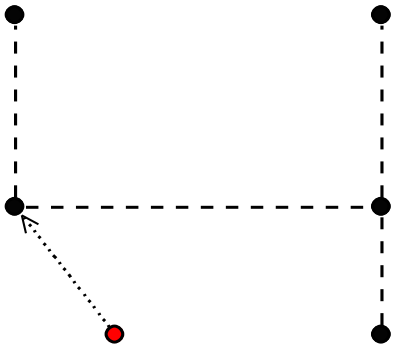
Las primeras mejoras que se deberían realizar son la implementación de movimientos de un lugar a otro, considerando lugar a cada las habitaciones y no a puntos concretos del modelo. Para ello habría que definir una serie de lugares para cada uno de los modelos, tal como ilustra el siguiente ejemplo.

Supongamos que se tiene un escenario como el siguiente:



Como se puede observar, dispone de cuatro salas, numeradas del 1 al 4. Se hace necesario establecer una serie de puntos a modo de puntos de estructura para cada sala. Estos puntos sirven de apoyo a la hora de obtener las rutas.

La idea parte de indicar a un personaje que debe ir, por ejemplo, a la sala 4, y suponiendo que el personaje parte desde cualquier punto de la sala 1 (punto rojo). Primero habrá que desplazar al personaje desde el punto en el que se encuentra al punto de control más cercano especificado en la estructura del modelo.



Una vez el personaje está situado en el punto de control bastará con que se desplace al punto de control más cercano al punto de control de destino, siempre y cuando esté unido al punto de control de partida. Aplicando algoritmos de por ejemplo el de Dijkstra, se puede obtener cual es el camino mínimo para ir de un punto origen al destino, obteniendo la ruta que se debe seguir para ir de un punto origen cualquiera, a un punto destino en una sala diferente a la de origen específico.

Otra de las mejoras que se plantean aplicar al servidor es la generación de modelos de entornos de manera automática. Hasta ahora, para que el sistema pueda renderizar un modelo hay que generarlo con una herramienta de modelado como puede ser 3D Studio (empleado para los modelos de este proyecto), la idea es que en lugar de emplear una herramienta para modelar un entorno se utilice la herramienta de modelado para diseñar las

piezas de las que se compondrá el modelo final. Mediante un fichero, ya sea de texto plano o XML (recomendado ya que permite anidamiento jerárquico).

Mediante esta definición jerárquica se puede construir un modelo que defina distintas plantas de un edificio, con sus distintas zonas, tal como se planteó en la mejora anterior. Mediante dicho fichero se puede indicar como será la planta baja, como será cada una de las plantas intermedias y cómo será la planta final, y el motor de render se encargará de colocar cada una de las piezas necesarias para componer cada una de las plantas y mostrar el resultado final. Gracias a esto, no es necesario definir cada una de las plantas, basta con definir la estructura de una de ellas y en el fichero definir cuantas plantas intermedias se desea tener.

A continuación se muestra un ejemplo de cómo podría ser el fichero, sin tener en cuenta las coordenadas en las que se situará cada pieza, solo se ilustra cual podría ser una definición de estructura de un fichero:

```
<modelo name="model1">
    <Baja>
        <pieza name ="pieza1">
            </pos x=x y=y z=z>
            </pos x=x y=y z=z>
            </pos x=x y=y z=z>
            ......
        </pieza>
        ......
    </Baja>
    <Resto>
        <pieza name ="piezaX">
            </pos x=x y=y z=z>
            </pos x=x y=y z=z>
            </pos x=x y=y z=z>
            ......
        </pieza>
        ......
    </Resto>
    <Estructura>
        </nBaja ammount=1>
        </nResto ammount=5>
    </Estructura>
</modelo>
```

Como se puede ver en el ejemplo, se definen las estructuras para una planta baja y para el resto de plantas, indicando las piezas que se emplearán en cada una de las plantas, etc…

Finalmente se define el número de plantas bajas, y el número de plantas "resto", partiendo de la base de que se van a generar en orden de lectura, es decir, primero se lee una planta baja, pues se realiza el renderizado de dicha planta baja, a continuación se lee cuantas plantas resto se desea tener y se renderizan las cinco plantas restantes y así sucesivamente.

7.2.2. Mejoras en los clientes

En cuanto a las mejoras en los futuros clientes que se puedan implementar podrían ser la adaptación del cliente generador de ficheros para soportar el movimiento por destinos, es decir, poder indicar a un personaje que vaya a la sala número 2, añadiendo a la interfaz de usuario una lista de posibles destinos y mediante la selección de uno de ellos y pulsando un botón, se envíe una orden para realizar el movimiento adecuado.

Otra de las mejoras que deberían implementarse para soportar las mejoras del servidor es añadir a la interfaz una lista de escenarios que contenga el servidor para poder indicar cual se desea cargar. En esta misma línea, se podría implementar un método para que en lugar de tener un fichero en el servidor, sea el cliente el que realice el envío de dicho fichero como si fuera una orden.

Adicionalmente se requerirán modificaciones tanto en el Web Service como en la librería libController y la librería Orden para soportar las nuevas órdenes que pueden ser enviadas.

Finalmente, podría ser interesante que los clientes no necesiten la intervención del usuario para generar un fichero o abrirlo, etc… La idea es que los clientes generen de manera aleatoria las distintas órdenes que se desean enviar al servidor. Esto permite dejar al "azar" (todo lo que el azar pueda influir en una máquina) para realizar acciones sobre los personajes. Pero mucho más interesante es la implementación de movimientos que respondan a modelos matemáticos acerca de las acciones que realizan personas en un entorno determinado.

Supongamos por ejemplo que se desea modelar una calle comercial y se desea simular el comportamiento de las personas que circulan por la calle. Bastará con indicar bajo qué modelo matemático responden y generar órdenes acordes a dicho modelo.

8. Anexos

8.1.El motor XNA

En este anexo se describe como XNA implementa los conceptos necesarios para la realización de un videojuego. Introduce algunos de los mecanismos que emplea para la carga y gestión de contenidos y como gestiona la ejecución de un juego.

8.1.1. Estructura general de un VideoJuego

La lógica central de un juego incluye la preparación del entorno donde el juego se lleva a cabo, ejecutar el juego en un bucle hasta que el juego termina, según los criterios de finalización del mismo y finalmente limpiar el entorno.

Es fundamental que la lógica principal de un juego se ejecute en un bucle, ya que el juego necesita mantenerse en ejecución halla o no interacción por parte del usuario, esto no ocurre con otro tipo de aplicaciones, en las que solo hay respuestas cuando el usuario realiza alguna acción.

Como ejemplo podemos ver el siguiente pseudocódigo simplificando la estructura de un juego:

Inicialización de gráficos, periféricos de entrada y dispositivos de sonido.
Cargar recursos (imágenes, archivos de audio, modelos 3D, etc…)
Iniciar el Bucle (en cada paso:)
    Actualizar la entrada por parte del usuario.
    Realizar cálculos (IA, movimientos, colisiones)
    Testeo del criterio de finalización.

Pintar en pantalla, generar sonidos, feedback al control del usuario.
Fin del bucle
Finalizar gráficos, sonidos y entrada.
Liberar recursos.

Esta es un poco la idea general de un juego, se podrían cargar recursos dentro del bucle, como pueden ser datos de nuevos niveles, enemigos, sonidos de acciones realizadas por el usuario, pero permite hacerse a la idea de cuál es la estructura general de cualquier juego.

Antes de la aparición de XNA, esta estructura la debía programar desde cero el programador, por lo que había que tener en cuenta muchos detalles que nada tenían que ver con el juego en sí. XNA oculta la mayoría de estos detalles.

En el siguiente pseudocódigo se puede ver como XNA implementa la estructura anterior:

Game1() – Inicialización general.
Initialize() – inicialización de componentes propios del juego.
LoadContent() – carga de recursos.
Run() – inicia el bucle.
Update() – lectura de la entrada, hacer cálculos y testeo de la finalización
Draw() – renderizado de imágenes.
UnloadContent () – liberar recursos.

8.1.2. Inicialización de un juego

La clase Game1, que es la que define el juego, comienza definiendo y creando objetos que referencian al controlador gráfico y un objeto "SpriteBatch" que se emplea para mostrar por pantalla texto e imágenes en 2D. El constructor de la clase Game1 también se encarga de configurar el directorio raíz para el gestor de contenidos, donde se almacenan los modelos, imágenes 2D, fuentes de texto, sonidos, etc… funcionando como punto de entrada para el Content Pipeline, informando al framewok de XNA de donde debe obtener los recursos que requiere.

8.1.2.1. El gestor de dispositivos gráficos

El gestor de dispositivos gráficos es el punto de entrada de la capa de manejo de gráficos, en incluye métodos, propiedades y eventos que permiten al programador consultar y modificar esta capa, es decir, representa la forma de manejar el acceso a las capacidades de la tarjeta gráfica.

Creando un objeto de la clase GraphicsDeviceManager, se crea una ventana en la que por medio del objeto creado para realizar cualquier operación grafica. Gracias a XNA, la complejidad acerca de la inicialización de la capa de gráficos, queda oculta para el programador.

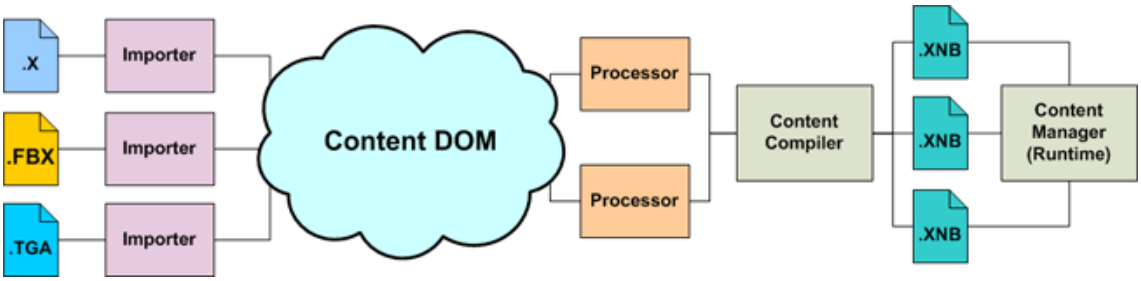
8.1.2.2. El Content Pipeline

El Content Pipeline es una de las funcionalidades más interesantes que ofrece XNA. Ayuda a simplificar como el juego interactúa con el contenido generado por las diferentes herramientas, como pueden ser editores de audio, modeladores 3D, etc…

En juegos que no están basados en el Framework de XNA, el programador debe preocuparse de cómo cargar el contenido de audio, gráficos, modelos 3D, en que directorio se encuentra el contenido, como el programa debe leer el contenido, asegurarse de que se disponen las librerías adecuadas, etc…

El Content Pipeline comprende un número de pasos, que incluyen sistemas de importación para leer el contenido y generarlo en un formato conocido, un lector para este formato y un compilador que genera contenido listo para su uso y finalmente el gestor de contenido.

La siguiente figura presenta una vista a alto nivel del Content Pipeline.



Una de las características más importantes del Content Pipeline es que está basando en contenido que el programador ha incluido en el proyecto, eso significa que cuando el proyecto es generado (build), el contenido es transformado en un formato reconocible y movido a un directorio concreto, con lo que el programa siempre sabrá dónde buscar el contenido y como leerlo.

8.1.2.3. Métodos de inicialización en XNA

Como se vio en el pseudocódigo en puntos anteriores, el bucle del juego necesita de una inicialización y carga de recursos. Esta carga inicial se realiza en los métodos Initialize y LoadContent.

El método Initialize es llamado cuando se ejecuta el método Run(), justo antes de iniciar el bucle principal. Este método es el adecuado para incluir cualquier inicialización no referente a los gráficos, como puede ser el contenido de audio, dispositivos de entrada, etc…

Así mismo, el método Initialize contiene una llamada al método Initialice de su superclase y éste a su vez itera sobre una colección de GameComponents, llamando al método intialize de cada uno de ellos. Esto permite que el programador pueda crear componentes que el juego utilizará. Esto facilita la reutilización de código y la modularidad de un juego, facilitando el mantenimiento de los mismos.

En el método LoadContent es donde se realizarla la carga de los gráficos, ya que en algunas ocasiones, puede ser necesario recargarlos. Los gráficos son cargados de acuerdo a los parámetros del dispositivo gráfico para obtener el máximo rendimiento. Como ejemplo si cambiamos la resolución de pantalla, habrá que notificárselo al dispositivo gráfico, y tras esto habrá que volver a cargar la parte gráfica volviendo a llamar al método LoadContent.

8.1.3. En resumen

En este anexo se han mostrado los conceptos principales presentes en un juego bastado en el Framework de XNA. Estos conceptos generales están presentes en cualquier

juego, por lo que fue muy importante su asimilación para la realización de este proyecto. Es importante comprenderlos bien antes de poder realizar cualquier proyecto con XNA.

También es importante recordar la estructura que tendrán los juegos de XNA y conocer los métodos de la clase que se puede sobrescribir para que el programador genere la lógica del juego.

8.2.Conceptos de transformaciones en 3 dimensiones

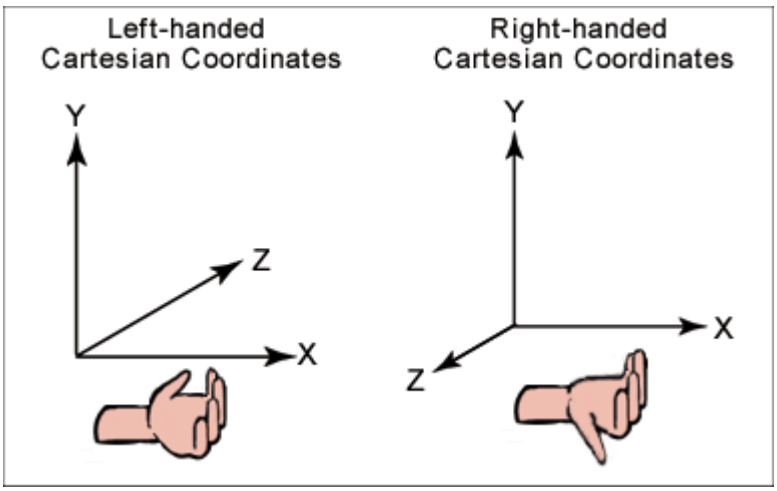
En este anexo se presentan los conceptos básicos que envuelve la creación de aplicaciones en 3D, concretamente los juegos, mostrando los conceptos necesarios para realizar transformaciones sobre objetos en 3D, sistemas de coordenadas, etc…

8.2.1. Sistemas de coordenadas en 3D y proyecciones

Cuando se habla de sistemas de coordenadas en 3 dimensiones, es necesario comprender algunos necesarios para la definición de objetos tridimensionales virtuales y de cómo se transforman en objetos bidimensionales en la pantalla. Primero se mostrarán estos conceptos básicos, posteriormente veremos cómo aplicarlos a la creación de un juego usando XNA.

Cuando se trabaja con sistemas Cartesianos de 3 dimensiones, tenemos dos tipos de sistemas, Left-handed (representados por la regla de la mano izquierda) y Right-handed (representados por la regla de la mano derecha. Estos nombres están referidos a la posición relativa del eje Z. En los sistemas Left-handed, los valores en el eje Z crecen a medida que se alejan del observador. En los sistemas Right-handed, los valores de Z crecen a medida que se acercan al observador. Considerando al observador como el usuario o la pantalla.

Por defecto XNA trabaja con un sistema de coordenadas de tipo right-handed.



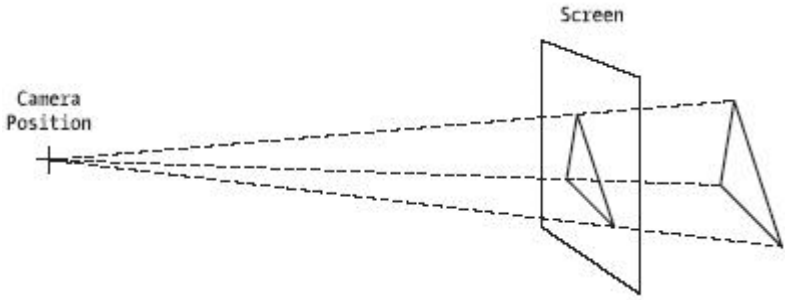
Una vez se conocen los dos sistemas de coordenadas cartesianas se mostrará la forma de proyectar los objetos 3D en la pantalla de ordenador, 2D.

Afortunadamente, XNA se encarga de realizar todo el trabajo matemático para mapear los objetos 3D en una proyección 2D. Pero es importante conocer los conceptos de proyecciones y como son aplicados por el motor de XNA.

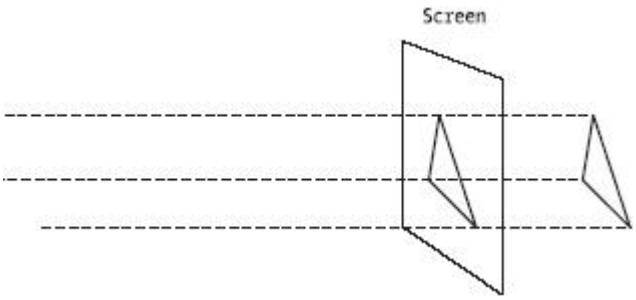
Xna soporta dos tipos de proyecciones:

- Proyección en perspectiva: es el tipo más común, toma en cuenta la distancia en el eje Z en la que se encuentra el objeto y ajusta el tamaño del mismo acorde a dicha distancia. En una proyección en perspectiva, los objetos más lejanos se verán más pequeños, mientras que los más cercanos se verán más grandes

- Proyección ortogonal: este tipo de proyección ignora la componente Z, el tamaño del objeto es siempre el mismo independientemente de su posición en el eje Z. Esta es la proyección más empleada en juegos 2D o en juegos 3D sencillos.



Proyección en perspectiva.



Proyección Ortogonal.

8.2.2. Vértices

La parte más básica de un objeto en 3 dimensiones es el Vertex o Vértice. Matemáticamente los vértices están representados por tres coordenadas, XNA mapea un vértice en un objeto de la clase Vector3. En sistemas de render 3D un vértice, además de sus

tres coordenadas tiene información adicional, como puede ser el color, textura, la normal del vector.

Cuando se crean objetos en 3 dimensiones, no nos basta con conocer la posición de los vértices y la información adicional, también es necesario especificar como XNA conectará dichos vértices de acuerdo a las distintas primitivas que existen.

Las primitivas indican a XNA como una colección de vértices serán renderizadas cuando las funciones de pintado sean invocadas. Estos podrán ser pintados como puntos sueltos, como un conjunto de líneas o como un conjunto de triángulos.

En este proyecto no se describen los tipos de primitivas ya que no son necesarias para la comprensión del mismo.

8.2.3. Vectores, matrices y tranformaciones en 3 dimensiones

Antes de poder empezar a crear la primera aplicación 3D, hay algunos conceptos que se deben comprender. Los más importantes son el uso de matrices y vectores. Es importante entender que los vectores, además de almacenar valores posicionales, proveen métodos para obtener la distancia entre dos puntos, sumar o restar vectores, multiplicarlos o dividirlos, calcular interpolaciones lineales entre dos vectores, etc... Así mismo las clases derivadas de la clase Vector, ofrecen vectores especiales como pueden ser los vectores de dirección Arriba (0,1,0), derecha(1,0,0) el vector nulo (0,0,0).

Las matrices son la base para definir rotaciones, escalados o traslaciones de un objeto en un entorno 3D. Debido a que las matrices son usadas para definir transformaciones en espacios tridimensionales, nos definen también las operaciones necesarias para simular las

proyecciones y para transformar una escena tridimensional de acuerdo a la posición de la cámara y a la orientación de la misma.

Como ejemplo, supongamos que se dispone de un triángulo, cuyos vértices están en las siguientes coordenadas según la tabla:

Vertex	**X**	**Y**	**Z**
1	50	10	0
2	70	10	0
3	55	25	0

Para trasladar el triángulo 40 unidades sobre el eje Y en la dirección positiva del eje, todo lo que hay que hacer es sumar a cada punto el vector v = (0,40,0), sin embargo, la forma canónica de hacerlo es extender el espacio de R3 a R4, de manera que todas las operaciones se realicen mediante multiplicaciones, por lo que cada uno de los puntos se representarán como una matriz de 1 fila y 4 columnas, finalmente bastará multiplicar cada vértice del triángulo por la matriz $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 40 & 0 & 1 \end{pmatrix}$ lo que producirá el desplazamiento de todos los puntos, una distancia de 40 unidades en el eje Y.

Tal como se puede observar, para realizar traslaciones, bastará con colocar las cantidades de la traslación en los valores X, Y, Z de la última fila de la matriz de transformación. Para realizar escalados se colocarán valores mayores que 1 (para ampliar) o fraccionarios (para reducir) en la diagonal principal, según en el eje en el que se desee reducir o ampliar, mientras que para realizar rotaciones bastará con colocar combinaciones de senos y cosenos en las posiciones específicas en la matriz.

El mayor beneficio de emplear matrices es que, como se mencionó anteriormente, todas las operaciones se pueden realizar a través de multiplicaciones, por complejas que sean dichas operaciones.

XNA abstrae todo el proceso matemático de generación de las matrices de transformación ofreciendo métodos que generan matrices de rotación, traslación escalado o matrices de visión, matrices para crear la perspectiva, etc…

8.3.Definiciones y acrónimos

- Escenario: nos referimos a escenario o entorno a un lugar que ha sido modelado mediante una herramienta de modelado en 3 Dimensiones, en el cual se moverán los personajes creados por los distintos clientes.
- Ethernet: Es un estándar de redes de área local. Define las características del cableado y señalización a nivel físico.
- HTTP (HyperText Transfer Protocol): es un protocolo usado en las transacciones que tienen que ver con la Web. Http define la sintaxis y la semántica que utilizan los elementos software de la arquitectura Web para comunicarse.
- IDE (Integrated Development Environment o Entorno de desarrollo integrado): es un programa informático compuesto por un conjunto de herramientas de programación. Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación. Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación diferente.
- Microsoft Zune: es el nombre que recibió el reproductor multimedia creado por Microsoft para competir con el conocido iPod de Appel. Zune no es solo un reproductor musical sino que también tiene soporte para ejecutar aplicaciones desarrolladas sobre la biblioteca .Net Compact Framework 3.0

- PDA (Personal Display Assistant): se denomina PDA a un pequeño ordenador de mano, con un diseño similar al de una agenda electrónica. Dispone de aplicaciones para gestión de agenda, contactos, notas y muchas otras que son instalables. En los últimos años estos dispositivos han mejorado enormemente, permitiendo su uso como teléfono móvil, navegador GPS, navegador de internet, o como reproductor multimedia.
- Puerto: en términos de una red de ordenadores, se denomina como puerto a una interfaz para comunicar un programa con otro a través de una red. Se numeran desde el 1 hasta el 65535, quedando reservados los primeros 1024 para el sistema operativo y protocolos conocidos. Los que están a partir del 49152 se denominan dinámicosya uqe son usados por el sistema operativo cuando una aplicación tiene que conectarse a un servidor.
- SOAP (Simple Object Access Protocol): protocolo creado por Microsoft, IBM y otros, que define como dos objetos en procesos diferentes, puede intercambiarse datos a través de XML.
- UDDI (*Universal Description, Discovery and Integration*): es uno de los estándares básicos de los Servicios Web, su objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los Servicios Web.
- WSDL WSDL (*Web Service Description Lenguaje*): lenguaje que describe la interfaz pública de un servicio web, este lenguaje está basado en XML.
- XBOX 360: es la segunda videoconsola producida por Microsoft. Gracias a XNA cualquier programador puede realizar juegos para la videoconsola y publicarlos para dejarlos disponibles en la comunidad Windows Live.
- XML (*eXtensible Markup Lenguaje*): lenguaje de marcas extensible es un lenguaje desarrollado por el *Word Wide Web Consortium* (W3C), es un lenguaje que permite definir lenguajes para diferentes necesidades.

9. Referencias

[1] ARCHER, Tom. *A fondo C#*.
McGraw-Hill Interamericana.

[2] Foros de soporte de Microsoft MSDN.
http://forums.microsoft.com/msdn-es/default.aspx?siteid=11

[3] W3C, Servicios Web.
http://www.w3c.es/Divulgacion/Guiasbreves/ServiciosWeb

[4] Wikipedia.
http://es.wikipedia.org

[5] Lobao, Evangelista, Leal de Farias. Beginning XNA 2.0 Game Programming.
APress. Http://www.apress.com

[6] Molero, Josep. 3ds Max 2009. Guía Rápida.
Inforbooks. http://www.inforbooks.com

[7] XNA Creators Club Online.
http://creators.xna.com/es-ES